Darcs User Manual

David Roundy

2.8.4 (+ 1 patch)

Contents

1 Introduction **4**
- 1.1 Features . . . 4

2 Getting started **6**
- 2.1 Creating your repository . . . 6
- 2.2 Making changes . . . 6
- 2.3 Making your repository visible to others . . . 7
- 2.4 Getting changes made to another repository . . . 7
- 2.5 Moving patches from one repository to another . . . 7
  - 2.5.1 Push . . . 8
  - 2.5.2 All pulls . . . 8
  - 2.5.3 Send and apply manually . . . 8

3 Configuring darcs **9**
- 3.1 prefs . . . 9
- 3.2 Environment variables . . . 12
- 3.3 General-purpose variables . . . 12
- 3.4 Remote repositories . . . 13
- 3.5 Highlighted output . . . 15
- 3.6 Character escaping and non-ASCII character encodings . . . 15

4 Best practices **16**
- 4.1 Creating patches . . . 16
  - 4.1.1 Changes . . . 16
  - 4.1.2 Keeping or discarding changes . . . 16
  - 4.1.3 Unrecording changes . . . 17
  - 4.1.4 Special patches and pending . . . 17
- 4.2 Using patches . . . 18
  - 4.2.1 Dependencies . . . 18
  - 4.2.2 Branches: just normal repositories . . . 18
  - 4.2.3 Moving patches around—no versions . . . 18
  - 4.2.4 Tags—versions . . . 19
  - 4.2.5 Conflicts . . . 19
  - 4.2.6 Resolving conflicts . . . 19
- 4.3 Global and per-repository caches . . . 20
- 4.4 Distributed development with one primary developer . . . 20
- 4.5 Sending signed patches by email . . . 21

5 Darcs commands **24**
5.1 Common options to darcs commands . . . 24
5.1.1 Match . . . 26
5.1.2 Posthooks . . . 30
5.1.3 Prehooks . . . 31
5.2 Options apart from darcs commands . . . 32
5.3 Getting help . . . 32
5.3.1 darcs help . . . 32
5.4 Creating repositories . . . 32
5.4.1 darcs initialize . . . 32
5.4.2 darcs get . . . 33
5.5 Modifying the contents of a repository . . . 34
5.5.1 darcs add . . . 34
5.5.2 darcs remove . . . 34
5.5.3 darcs move . . . 35
5.5.4 darcs replace . . . 35
5.6 Working with changes . . . 36
5.6.1 darcs record . . . 36
5.6.2 darcs pull . . . 38
5.6.3 darcs push . . . 41
5.6.4 darcs send . . . 43
5.6.5 darcs apply . . . 45
5.7 Seeing what you've done . . . 48
5.7.1 darcs whatsnew . . . 48
5.7.2 darcs changes . . . 48
5.8 More advanced commands . . . 49
5.8.1 darcs tag . . . 49
5.8.2 darcs setpref . . . 50
5.8.3 darcs test . . . 50
5.8.4 darcs repair . . . 51
5.8.5 darcs optimize . . . 51
5.9 Undoing, redoing and running in circles . . . 52
5.9.1 darcs amend-record . . . 52
5.9.2 darcs rollback . . . 53
5.9.3 darcs unrecord . . . 54
5.9.4 darcs obliterate . . . 56
5.9.5 darcs revert . . . 57
5.9.6 darcs unrevert . . . 58
5.10 Advanced examination of the repository . . . 58
5.10.1 darcs diff . . . 58
5.10.2 darcs annotate . . . 59
5.10.3 darcs show . . . 60
5.11 Rarely needed and obscure commands . . . 62
5.11.1 darcs mark-conflicts . . . 62
5.11.2 darcs trackdown . . . 63
5.11.3 darcs dist . . . 64
5.11.4 darcs put . . . 64
5.11.5 darcs fetch . . . 65
5.11.6 darcs convert . . . 66

A Building darcs **67**

B License **68**

Chapter 1

Introduction

This manual provides a stable documentation for using darcs. To find more up-to-date and complementary information, please consult the darcs wiki[1].

Darcs is a revision control system, along the lines of Subversion, Git or Mercurial. That means that it keeps track of various revisions and branches of your project, allows for changes to propagate from one branch to another. Darcs has two particularly distinctive features which differ from other revision control systems:

Every source tree a branch The primary simplifying notion of darcs is that *every* copy of your source code is a full repository. This is dramatically different from Subversion, in which the normal usage is for there to be one central repository from which source code will be checked out. This has several advantages, since you can harness the full power of darcs in any scratch copy of your code, without committing your possibly destabilizing changes to a central repository.

Darcs keeps track of changes rather than versions In the world of darcs, the source tree is *not* the fundamental object, but rather the patch is the fundamental object. Rather than a patch being defined in terms of the difference between two trees, a tree is defined as the result of applying a given set of patches to an empty tree. Moreover, these patches may be reordered without changing the tree. This makes many operations, like cherry-picking or merging, much more natural.

1.1 Features

Record changes locally In darcs, the equivalent of a svn "commit" is called record, because it doesn't put the change into any remote or centralized repository. Changes are always recorded locally, meaning no net access is required in order to work on your project and record changes as you make them.

Interactive records You can choose to perform an interactive record, in which case darcs will prompt you for each change you have made and ask if you wish to record it. Of course, you can tell darcs to record all the changes in a given file, or to skip all the changes in a given file, or go back to a previous change, or whatever.

Unrecord local changes As a corollary to the "local" nature of the record operation, if a change hasn't yet been published to the world—that is, if the local repository isn't accessible by others—you can safely unrecord a change (even if it wasn't the most recently recorded change) and then re-record it differently, for example if you forgot to add a file, introduced a bug or realized that what you recorded as a single change was really two separate changes.

[1] http://wiki.darcs.net

Interactive everything else Most darcs commands support an interactive interface. The "revert" command, for example, which undoes unrecorded changes has the same interface as record, so you can easily revert just a single change. Pull, push, send and apply all allow you to view and interactively select which changes you wish to pull, push, send or apply.

Test suites Darcs has support for integrating a test suite with a repository. If you choose to use this, you can define a test command (e.g. "make check") and have darcs run that command on a clean copy of the project either prior to recording a change or prior to applying changes—and to reject changes that cause the test to fail.

Any old server Darcs does not require a specialized server in order to make a repository available for read access. You can use http, ftp, or even just a plain old ssh server to access your darcs repository.

You decide write permissions Darcs doesn't try to manage write access. That's your business. Supported push methods include direct ssh access (if you're willing to *give* direct ssh access away), using sudo to allow users who already have shell access to only apply changes to the repository, or verification of gpg-signed changes sent by email against a list of allowed keys. In addition, there is support for submission of patches by email that are not automatically applied, but can easily be applied with a shell escape from a mail reader.

File and directory moves Renames or moves of files and directories, of course are handled properly, so when you rename a file or move it to a different directory, its history is unbroken, and merges with repositories that don't have the file renamed will work as expected.

Token replace You can use the "darcs replace" command to modify all occurrences of a particular token (defined by a configurable set of characters that are allowed in "tokens") in a file. This has the advantage that merges with changes that introduce new copies of the old token will have the effect of changing it to the new token—which comes in handy when changing a variable or function name that is used throughout a project.

Configurable defaults You can easily configure the default flags passed to any command on either a per-repository or a per-user basis or a combination thereof.

Chapter 2

Getting started

This chapter will lead you through an example use of darcs, which hopefully will allow you to get started using darcs with your project.

2.1 Creating your repository

Creating your repository in the first place just involves telling darcs to create the special directory (called `_darcs`) in your project tree, which will hold the revision information. This is done by simply calling from the root directory of your project:

```
$ cd my_project/
$ darcs initialize
```

This creates the `_darcs` directory and populates it with whatever files and directories are needed to describe an empty project. You now need to tell darcs what files and directories in your project should be under revision control. You do this using the command `darcs add`:

```
$ darcs add *.c Makefile.am configure.ac
```

When you have added all your files (or at least, think you have), you will want to record your changes. "Recording" always includes adding a note as to why the change was made, or what it does. In this case, we'll just note that this is the initial version.

```
$ darcs record --all
What is the patch name? Initial revision.
```

Note that since we didn't specify a patch name on the command line we were prompted for one. If the environment variable 'EMAIL' isn't set, you will also be prompted for your email address. Each patch that is recorded is given a unique identifier consisting of the patch name, its creator's email address, the date when it was created, and a random hash.

2.2 Making changes

Now that we have created our repository, make a change to one or more of your files. After making the modification run:

```
$ darcs whatsnew
```

This should show you the modifications that you just made, in the darcs patch format. If you prefer to see your changes in a different format, read Section 5.7.1, which describes the whatsnew command in detail.

Let's say you have now made a change to your project. The next thing to do is to record a patch. Recording a patch consists of grouping together a set of related changes, and giving them a name. It also tags the patch with the date it was recorded and your email address.

To record a patch simply type:

```
$ darcs record
```

darcs will then prompt you with all the changes that you have made that have not yet been recorded, asking you which ones you want to include in the new patch. Finally, darcs will ask you for a name for the patch.

You can now rerun whatsnew, and see that indeed the changes you have recorded are no longer marked as new.

2.3 Making your repository visible to others

How do you let the world know about these wonderful changes? Obviously, they must be able to see your repository. Currently the easiest way to do this is is to look for hosting services[1] for darcs repositories.

You can also self-host your repositories by using any web server. The recommended way to do this (using apache in a UNIX environment) is to create a directory called `/var/www/repos`, and then put a symlink to your repository there:

```
$ cd /var/www/repos
$ ln -s /home/username/myproject .
```

2.4 Getting changes made to another repository

Ok, so your repository is browsable using any web browser... so what? How does one get your changes into *their* repository, where they can do some good? It couldn't be easier. One just `cd` into their repository, and there type:

```
$ darcs pull http://your.server.org/repos/yourproject
```

Darcs will check to see if you have recorded any changes that aren't in their current repository. If so, it'll prompt them for each one, to see which ones one want to add to their repository. Note that you may see a different series of prompts depending on your answers, since sometimes one patch depends on another, so if you answer yes to the first one, you won't be prompted for the second if the first depends on it.

Of course, maybe people don't even have a copy of your repository. In that case they'd want to do a

```
$ darcs get http://your.server.org/repos/yourproject
```

which gets the whole repository.

Get, pull and push also work over ssh. Ssh-paths are of the same form accepted by scp, namely `[username@]host:/path/to/repository`.

2.5 Moving patches from one repository to another

Darcs is flexible as to how you move patches from one repository to another. This section will introduce all the ways you can get patches from one place to another, starting with the simplest and moving to the most complicated.

[1] http://wiki.darcs.net/Hosting

2.5.1 Push

If you use ssh, you can use the push method to transfer changes. Push can also be used when the target repository is local, in which case ssh isn't needed.

Note that you can use push to administer a multiple-user repository. You just need to create a user for the repository (or repositories), and give everyone with write access ssh access, perhaps using `.ssh/authorized_keys`. Then they run

```
$ darcs push repouser@repo.server:repo/directory
```

If you like this idea about creating a repository user to own a repository which is writable by a number of users, you have one other option.

Push `--apply-as` can run on either a local repository or one accessed with ssh, but uses `sudo` to run a darcs apply command (having created a patch bundle as in send) as another user. You can add the following line in your `sudoers` file to allow the users to apply their patches to a centralized repository:

```
ALL   ALL = (repo-user) NOPASSWD: /usr/bin/darcs apply --all --repodir /repo/path*
```

This method is ideal for a centralized repository when all the users have accounts on the same computer, if you don't want your users to be able to run arbitrary commands as repo-user.

2.5.2 All pulls

This method involves making each repository readable (by http, ftp, nfs-mounted disk, or a public web hosting), and you run `darcs pull` in the repository you want to move the patch to. This is nice, as it doesn't require you to give write access to anyone else, and is reasonably simple.

2.5.3 Send and apply manually

Sometimes the push method is impossible because the owner of the main repository does not want or cannot create a user to write into it, and you cannot use the all-pull method because you cannot set up a web server on your machine, perhaps because it's behind a firewall or perhaps for security reasons, or because it is often turned off. In this case you can use `darcs send` from that computer to generate a patch bundle for the remote repository. You can either let darcs email the patch for you, or save it as a file and transfer it by hand. Then in the destination repository the owner of that repository runs `darcs apply` to apply the patches contained in the bundle. This is also quite a simple method since, like the all-pull method, it doesn't require that you give anyone write access to your repository. But it's less convenient, since the owner of the remote repository has to keep track of the patch bundle (in the email, or whatever).

To use the send and apply method with email, the best is probably to create a `_darcs/prefs/email` file in the target repository containing the email address of the maintainer. This way anyone who sends to this repository will automatically send the patch bundle to that email address.

If you receive many patches by email, you probably will benefit by running darcs apply directly from your mail program. You can place in your `.muttrc` the following:

```
auto_view text/x-patch text/x-darcs-patch
macro pager A "<pipe-entry>darcs apply --verbose --mark-conflicts \
        --reply droundy@abridgegame.org --repodir ~/darcs"
```

which will allow you to view a sent patch, and then apply the patch directly from `mutt`, sending a confirmation email to the person who sent you the patch. The autoview line relies on on the following lines, or something like them, being present in one's `.mailcap`:

```
text/x-patch;                           cat; copiousoutput
text/x-darcs-patch;                     cat; copiousoutput
```

Chapter 3

Configuring darcs

There are several ways you can adjust darcs' behavior to suit your needs. The first is to edit files in the `_darcs/prefs/` directory of a repository. Such configuration only applies when working with that repository. To configure darcs on a per-user rather than per-repository basis (but with essentially the same methods), you can edit (or create) files in the `~/.darcs/` directory. Finally, the behavior of some darcs commands can be modified by setting appropriate environment variables.

Microsoft Windows The global darcs directory is `%APPDATA%\darcs\`. This typically expands to `C:\Documents And Settings\`*user*`\Application Data\darcs\`. This folder contains the cache, as well as all the per-user settings files: preferences, boring etc... These will became the new defaults that can be overridden on per-repository basis.

3.1 prefs

The `_darcs` directory contains a `prefs` directory. This directory exists simply to hold user configuration settings specific to this repository. The contents of this directory are intended to be modifiable by the user, although in some cases a mistake in such a modification may cause darcs to behave strangely.

defaults Default values for darcs commands can be configured on a per-repository basis by editing (and possibly creating) the `_darcs/prefs/defaults` file. Each line of this file has the following form:

```
COMMAND FLAG VALUE
```

where `COMMAND` is either the name of the command to which the default applies, or `ALL` to indicate that the default applies to all commands accepting that flag. The `FLAG` term is the name of the long argument option without the "`--`", i.e. `verbose` rather than `--verbose`. Finally, the `VALUE` option can be omitted if the flag is one such as `verbose` that doesn't involve a value. If the value has spaces in it, use single quotes, not double quotes, to surround it. Each line only takes one flag. To set multiple defaults for the same command (or for `ALL` commands), use multiple lines.

Note that the use of `ALL` easily can have unpredicted consequences, especially if commands in newer versions of darcs accepts flags that they didn't in previous versions. A command like `obliterate` could be devastating with the "wrong" flags (for example –all). Only use safe flags with `ALL`.

`~/.darcs/defaults`	provides defaults for this user account (for MS Windows, see 3)
`repo/_darcs/prefs/defaults`	provides defaults for one project, overrules changes per user

For example, if your system clock is bizarre, you could instruct darcs to always ignore the file modification times by adding the following line to your `_darcs/prefs/defaults` file. (Note that this would have to be done for each repository!)

```
ALL ignore-times
```

If you never want to run a test when recording to a particular repository (but still want to do so when running `check` on that repository), and like to name all your patches "Stupid patch", you could use the following:

```
record no-test
record patch-name Stupid patch
```

If you would like a command to be run every time patches are recorded in a particular repository (for example if you have one central repository, that all developers contribute to), then you can set apply to always run a command when apply is successful. For example, if you need to make sure that the files in the repository have the correct access rights you might use the following. There are two things to note about using darcs this way:

- Without the second line you will get errors, because the sub process that runs apply cannot prompt interactively.
- Whatever script is run by the post apply command should not be added to the repository with `darcs add`; doing so would allow people to modify that file and then run arbitrary scripts on your main repository, possibly damaging or violating security.

```
apply posthook chmod -R a+r *
apply run-posthook
```

Similarly, if you need a command to run automatically before darcs performs an action you can use a prehook. Using prehooks it could be possible to canonicalize line endings before recording patches.

There are some options which are meant specifically for use in `_darcs/prefs/defaults`. One of them is `--disable`. As the name suggests, this option will disable every command that got it as argument. So, if you are afraid that you could damage your repositories by inadvertent use of a command like amend-record, add the following line to `_darcs/prefs/defaults`:

```
amend-record disable
```

Also, a global preferences file can be created with the name `.darcs/defaults` in your home directory, on MS Windows 3. Options present there will be added to the repository-specific preferences. If they conflict with repository-specific options, the repository-specific ones will take precedence.

repos The `_darcs/prefs/repos` file contains a list of repositories you have pulled from or pushed to, and is used for autocompletion of pull and push commands in bash. Feel free to delete any lines from this list that might get in there, or to delete the file as a whole.

author The `_darcs/prefs/author` file contains the email address (or name) to be used as the author when patches are recorded in this repository, e.g. `David Roundy <droundy@abridgegame.org>`. This file overrides the contents of the environment variables `$DARCS_EMAIL` and `$EMAIL`.

boring The `_darcs/prefs/boring` file may contain a list of regular expressions describing files, such as object files, that you do not expect to add to your project. As an example, you could have:

```
\.hi$
\.o$
^\.[^/]
^_
~$
(^|/)CVS($|/)
```

A newly created repository has a longer boring file that includes many common source control, backup, temporary, and compiled files.

You may want to have the boring file under version control. To do this you can use darcs setpref to set the value "boringfile" to the name of your desired boring file (e.g. `darcs setpref boringfile .boring`, where `.boring` is the repository path of a file that has been darcs added to your repository). The boringfile preference overrides `_darcs/prefs/boring`, so be sure to copy that file to the boringfile.

You can also set up a "boring" regexps file in your home directory, named `~/.darcs/boring`, (see 3 on MS Windows), which will be used with all of your darcs repositories.

Any file not already managed by darcs and whose repository path (such as `manual/index.html`) matches any of the boring regular expressions is considered boring. The boring file is used to filter the files provided to darcs add, to allow you to use a simple `darcs add newdir newdir/*` without accidentally adding a bunch of object files. It is also used when the `--look-for-adds` flag is given to whatsnew or record. Note that once a file has been added to darcs, it is not considered boring, even if it matches the boring file filter.

binaries The `_darcs/prefs/binaries` file may contain a list of regular expressions describing files that should be treated as binary files rather than text files. Darcs automatically treats files containing `^Z\` or `'\0'` within the first 4096 bytes as being binary files. You probably will want to have the binaries file under version control. To do this you can use darcs setpref to set the value "binariesfile" to the name of your desired binaries file (e.g. `darcs setpref binariesfile ./.binaries`, where `.binaries` is a file that has been darcs added to your repository). As with the boring file, you can also set up a `~/.darcs/binaries` file if you like (see 3 on MS Windows).

email The `_darcs/prefs/email` file is used to provide the e-mail address for your repository that others will use when they `darcs send` a patch back to you. The contents of the file should simply be an e-mail address.

sources The `_darcs/prefs/sources` file is used to indicate alternative locations from which to download patches when using a "hashed" repository. This file contains lines such as:

```
cache:/home/droundy/.darcs/cache
readonly:/home/otheruser/.darcs/cache
repo:http://darcs.net
```

This would indicate that darcs should first look in `/home/droundy/.darcs/cache` for patches that might be missing, and if the patch isn't there, it should save a copy there for future use. In that case, darcs will look in `/home/otheruser/.darcs/cache` to see if that user might have downloaded a copy, but won't try to save a copy there, of course. Finally, it will look in `http://darcs.net`. Note that the `sources` file can also exist in `~/.darcs/`. Also note that the sources mentioned in your `sources` file will be tried *before* the repository you are pulling from. This can be useful in avoiding downloading patches multiple times when you pull from a remote repository to more than one local repository.

A global cache is enabled by default in your home directory. The cache allows darcs to avoid re-downloading patches (for example, when doing a second darcs get of the same repository), and also allows darcs to use hard links to reduce disk usage.

Note that the cache directory should reside on the same filesystem as your repositories, so you may need to vary this. You can also use multiple cache directories on different filesystems, if you have several filesystems on which you use darcs.

motd The `_darcs/prefs/motd` file may contain a "message of the day" which will be displayed to users who get or pull from the repository without the `--quiet` option.

3.2 Environment variables

There are a few environment variables whose contents affect darcs' behavior. Here is a quick list of all the variables and their documentation in the rest of the manual:

Variable	Section
DARCS_EDITOR, EDITOR, VISUAL	3.3
DARCS_PAGER, PAGER	3.3
HOME	3.3
TERM	??
DARCS_EMAIL, EMAIL	5.1.1
DARCS_APPLY_FOO	3.4
DARCS_GET_FOO	3.4
DARCS_MGET_FOO	3.4
DARCS_MGETMAX	3.4
DARCS_PROXYUSERPWD	3.4
DARCS_CONNECTION_TIMEOUT	3.4
DARCS_SSH	3.4
DARCS_SCP	3.4
DARCS_SFTP	??
SSH_PORT	3.4
DARCS_ALTERNATIVE_COLOR	3.5
DARCS_ALWAYS_COLOR	3.5
DARCS_DO_COLOR_LINES	3.5
DARCS_DONT_COLOR	3.5
DARCS_DONT_ESCAPE_TRAILING_CR	3.5
DARCS_DONT_ESCAPE_TRAILING_SPACES	3.5
DARCS_DONT_ESCAPE_8BIT	3.6
DARCS_DONT_ESCAPE_ANYTHING	3.6
DARCS_DONT_ESCAPE_ISPRINT	3.6
DARCS_ESCAPE_EXTRA	3.6
DARCS_DONT_ESCAPE_EXTRA	3.6

3.3 General-purpose variables

DARCS_EDITOR, DARCSEDITOR, VISUAL and EDITOR To edit a patch description of email comment, Darcs will invoke an external editor. Your preferred editor can be set as any of the environment variables $DARCS_EDITOR, $DARCSEDITOR, $VISUAL or $EDITOR. If none of these are set, vi(1) is used. If vi crashes or is not found in your PATH, emacs, emacs -nw, nano and (on Windows) edit are each tried in turn.

DARCS_PAGER and PAGER Darcs will sometimes invoke a pager if it deems output to be too long to fit onscreen. Darcs will use the pager specified by $DARCS_PAGER or $PAGER. If neither are set, 'less' will be used.

DARCS_TMPDIR and TMPDIR Darcs often creates temporary directories. For example, the 'darcs diff' command creates two for the working trees to be diffed. By default temporary directories are created in /tmp, or if that doesn't exist, in _darcs (within the current repo). This can be overridden by specifying some other directory in the file _darcs/prefs/tmpdir or the environment variable $DARCS_TMPDIR or $TMPDIR.

DARCS_KEEP_TMPDIR If the environment variable DARCS_KEEP_TMPDIR is defined, darcs will not remove the temporary directories it creates. This is intended primarily for debugging Darcs itself, but

it can also be useful, for example, to determine why your test preference (see 'darcs setpref') is failing when you run 'darcs record', but working when run manually.

HOME and APPDATA Per-user preferences are set in $HOME/.darcs (on Unix) or %APPDATA%/darcs (on Windows). This is also the default location of the cache.

3.4 Remote repositories

DARCS_CONNECTION_TIMEOUT Set the maximum time in seconds that darcs allows and connection to take. If the variable is not specified the default are 30 seconds. This option only works with curl.

DARCS_SSH Repositories of the form [user@]host:[dir] are taken to be remote repositories, which Darcs accesses with the external program ssh(1).

The environment variable $DARCS_SSH can be used to specify an alternative SSH client. Arguments may be included, separated by whitespace. The value is not interpreted by a shell, so shell constructs cannot be used; in particular, it is not possible for the program name to contain whitespace by using quoting or escaping.

DARCS_SCP and DARCS_SFTP When reading from a remote repository, Darcs will attempt to run 'darcs transfer-mode' on the remote host. This will fail if the remote host only has Darcs 1 installed, doesn't have Darcs installed at all, or only allows SFTP.

If transfer-mode fails, Darcs will fall back on scp(1) and sftp(1). The commands invoked can be customized with the environment variables $DARCS_SCP and $DARCS_SFTP respectively, which behave like $DARCS_SSH. If the remote end allows only sftp, try setting DARCS_SCP=sftp.

SSH_PORT If this environment variable is set, it will be used as the port number for all SSH calls made by Darcs (when accessing remote repositories over SSH). This is useful if your SSH server does not run on the default port, and your SSH client does not support ssh_config(5). OpenSSH users will probably prefer to put something like 'Host *.example.net Port 443' into their /.ssh/config file.

HTTP_PROXY, HTTPS_PROXY, FTP_PROXY, ALL_PROXY and NO_PROXY If Darcs was built with libcurl, the environment variables HTTP_PROXY, HTTPS_PROXY and FTP_PROXY can be set to the URL of a proxy in the form

[protocol://]¡host¿[:port]

In which case libcurl will use the proxy for the associated protocol (HTTP, HTTPS and FTP). The environment variable ALL_PROXY can be used to set a single proxy for all libcurl requests.

If the environment variable NO_PROXY is a comma-separated list of host names, access to those hosts will bypass proxies defined by the above variables. For example, it is quite common to avoid proxying requests to machines on the local network with

NO_PROXY=localhost,*.localdomain

For compatibility with lynx et al, lowercase equivalents of these environment variables (e.g. $http_proxy) are also understood and are used in preference to the uppercase versions.

If Darcs was not built with libcurl, all these environment variables are silently ignored, and there is no way to use a web proxy.

DARCS_PROXYUSERPWD If Darcs was built with libcurl, and you are using a web proxy that requires authentication, you can set the $DARCS_PROXYUSERPWD environment variable to the username and password expected by the proxy, separated by a colon. This environment variable is silently ignored if Darcs was not built with libcurl.

DARCS_GET_FOO, DARCS_MGET_FOO and DARCS_APPLY_FOO When trying to access a repository with a URL beginning foo://, darcs will invoke the program specified by the DARCS_GET_FOO environment variable (if defined) to download each file, and the command specified by the DARCS_APPLY_FOO environment variable (if defined) when pushing to a foo:// URL.

This method overrides all other ways of getting `foo://xxx` URLs.

Note that each command should be constructed so that it sends the downloaded content to STDOUT, and the next argument to it should be the URL. Here are some examples that should work for DARCS_GET_HTTP:

```
fetch -q -o -
curl -s -f
lynx -source
wget -q -O -
```

Apart from such toy examples, it is likely that you will need to manipulate the argument before passing it to the actual fetcher program. For example, consider the problem of getting read access to a repository on a CIFS (SMB) share without mount privileges:

```
export DARCS_GET_SMB="smbclient -c get"
darcs get smb://fs/twb/Desktop/hello-world
```

The above command will not work for several reasons. Firstly, Darcs will pass it an argument beginning with 'smb:', which smbclient does not understand. Secondly, the host and share '//fs/twb' must be presented as a separate argument to the path 'Desktop/hello-world'. Thirdly, smbclient requires that 'get' and the path be a single argument (including a space), rather than two separate arguments. Finally, smbclient's 'get' command writes the file to disk, while Darcs expects it to be printed to standard output.

In principle, we could get around such problems by making the variable contain a shell script, e.g.

```
export DARCS_GET_SMB='sh -c "...; smbclient $x -c \"get $y\""'
```

Unfortunately, Darcs splits the command on whitespace and does not understand that quotation or escaping, so there is no way to make Darcs pass the text after '-c' to sh as a single argument. Therefore, we instead need to put such one-liners in separate, executable scripts.

Continuing our smbclient example, we create an executable script `~/.darcs/libexec/get_smb` with the following contents:

```
#!/bin/bash -e
IFS=/ read host share file <<<"${1#smb://}"
smbclient //$host/$share -c "get $file -"
```

And at last we can say

```
export DARCS_GET_SMB=~/.darcs/libexec/get_smb
darcs get smb://fs/twb/Desktop/hello-world
```

If set, DARCS_MGET_FOO will be used to fetch many files from a single repository simultaneously. Replace FOO and foo as appropriate to handle other URL schemes. These commands are *not* interpreted by a shell, so you cannot use shell metacharacters, and the first word in the command must be the name of an executable located in your path. The GET command will be called with a URL for each file. The MGET command will be invoked with a number of URLs and is expected to download the files to the current directory, preserving the file name but not the path. The APPLY command will be called with a darcs patchfile piped into its standard input. Example:

```
wget -q
```

DARCS_MGETMAX When invoking a DARCS_MGET_FOO command, darcs will limit the number of URLs presented to the command to the value of this variable, if set, or 200.

These commands are *not* interpreted by a shell, so you cannot use shell meta-characters.

3.5 Highlighted output

If the terminal understands ANSI color escape sequences, darcs will highlight certain keywords and delimiters when printing patches. This can be turned off by setting the environment variable DARCS_DONT_COLOR to 1. If you use a pager that happens to understand ANSI colors, like `less -R`, darcs can be forced always to highlight the output by setting DARCS_ALWAYS_COLOR to 1. If you can't see colors you can set DARCS_ALTERNATIVE_COLOR to 1, and darcs will use ANSI codes for bold and reverse video instead of colors. In addition, there is an extra-colorful mode, which is not enabled by default, which can be activated with DARCS_DO_COLOR_LINES.

By default darcs will escape (by highlighting if possible) any kind of spaces at the end of lines when showing patch contents. If you don't want this you can turn it off by setting DARCS_DONT_ESCAPE_TRAILING_SPACES to 1. A special case exists for only carriage returns: DARCS_DONT_ESCAPE_TRAILING_CR.

3.6 Character escaping and non-ASCII character encodings

Darcs needs to escape certain characters when printing patch contents to a terminal. Characters like *backspace* can otherwise hide patch content from the user, and other character sequences can even in some cases redirect commands to the shell if the terminal allows it.

By default darcs will only allow printable 7-bit ASCII characters (including space), and the two control characters *tab* and *newline*. (See the last paragraph in this section for a way to tailor this behavior.) All other octets are printed in quoted form (as `^<control letter>` or `\<hex code>`).

Darcs has some limited support for locales. If the system's locale is a single-byte character encoding, like the Latin encodings, you can set the environment variable DARCS_DONT_ESCAPE_ISPRINT to 1 and darcs will display all the printables in the current system locale instead of just the ASCII ones. NOTE: This curently does not work on some architectures if darcs is compiled with GHC 6.4 or later. Some non-ASCII control characters might be printed and can possibly spoof the terminal.

For multi-byte character encodings things are less smooth. UTF-8 will work if you set DARCS_DONT_ESCAPE_8BIT to 1, but non-printables outside the 7-bit ASCII range are no longer escaped. E.g., the extra control characters from Latin-1 might leave your terminal at the mercy of the patch contents. Space characters outside the 7-bit ASCII range are no longer recognized and will not be properly escaped at line endings.

As a last resort you can set DARCS_DONT_ESCAPE_ANYTHING to 1. Then everything that doesn't flip code sets should work, and so will all the bells and whistles in your terminal. This environment variable can also be handy if you pipe the output to a pager or external filter that knows better than darcs how to handle your encoding. Note that *all* escaping, including the special escaping of any line ending spaces, will be turned off by this setting.

There are two environment variables you can set to explicitly tell darcs to not escape or escape octets. They are DARCS_DONT_ESCAPE_EXTRA and DARCS_ESCAPE_EXTRA. Their values should be strings consisting of the verbatim octets in question. The do-escapes take precedence over the dont-escapes. Space characters are still escaped at line endings though. The special environment variable DARCS_DONT_ESCAPE_TRAILING_CR turns off escaping of carriage return last on the line (DOS style).

Chapter 4

Best practices

This chapter is intended to review various scenarios and describe in each case effective ways of using darcs. There is no one "best practice", and darcs is a sufficiently low-level tool that there are many high-level ways one can use it, which can be confusing to new users.

4.1 Creating patches

This section will lay down the concepts around patch creation. The aim is to develop a way of thinking that corresponds well to how darcs is behaving — even in complicated situations.

In a single darcs repository you can think of two "versions" of the source tree. They are called the *working* and *pristine* trees. *Working* is your normal source tree, with or without darcs alongside. The only thing that makes it part of a darcs repository is the `_darcs` directory in its root. *Pristine* is the recorded state of the source tree. The pristine tree is constructed from groups of changes, called *patches* (some other version control systems use the term *changeset* instead of *patch*).[1] Darcs will create and store these patches based on the changes you make in *working*.

4.1.1 Changes

If *working* and *pristine* are the same, there are "no changes" in the working copy. Changes can be introduced (or removed) by editing the files in *working*. They can also be caused by darcs commands, which can modify *both working* and *pristine*. It is important to understand for each darcs command how it modifies *working*, *pristine* or both of them.

`whatsnew` (as well as `diff`) can show the difference between *working* and *pristine* to you. It will be shown as a difference in *working*. In advanced cases it need *not* be *working* that has changed; it can just as well have been *pristine*, or both. The important thing is the difference and what darcs can do with it.

4.1.2 Keeping or discarding changes

If you have a difference in *working*, you do two things with it: `record` it to keep it, or `revert` it to lose the changes.[2]

If you have a difference between *working* and *pristine*—for example after editing some files in *working*—`whatsnew` will show some "unrecorded changes". To save these changes, use `record`. It will create a new patch in *pristine* with the same changes, so *working* and *pristine* are no longer different. To instead undo

[1] If you look inside _darcs you will find files or directories named `patches` and `inventories`, which store all the patches ever recorded. If the repository holds a cached pristine tree, it is stored in a directory called `pristine.hashed`.

[2] Revert can undo precious work in a blink. To protect you from great grief, the discarded changes are saved temporarily so the latest revert can be undone with unrevert.

the changes in *working*, use `revert`. It will modify the files in *working* to be the same as in *pristine* (where the changes do not exist).

4.1.3 Unrecording changes

`unrecord` is a command meant to be run only in private repositories. Its intended purpose is to allow developers the flexibility to undo patches that haven't been distributed yet.

If you `unrecord` a patch, that patch will be deleted from *pristine*. This will cause *working* to be different from *pristine*, and `whatsnew` to report unrecorded changes. The difference will be the same as just before that patch was `record`ed. Think about it. `record` examines what's different with *working* and constructs a patch with the same changes in *pristine* so they are no longer different. `unrecord` deletes this patch; the changes in *pristine* disappear and the difference is back.

If the recorded changes included an error, the resulting flawed patch can be unrecorded. When the changes have been fixed, they can be recorded again as a new—hopefully flawless—patch.

If the whole change was wrong it can be discarded from *working* too, with `revert`. `revert` will update *working* to the state of *pristine*, in which the changes do no longer exist after the patch was deleted.

Keep in mind that the patches are your history, so deleting them with `unrecord` makes it impossible to track what changes you *really* made. Redoing the patches is how you "cover the tracks". On the other hand, it can be a very convenient way to manage and organize changes while you try them out in your private repository. When all is ready for shipping, the changes can be reorganized in what seems as useful and impressive patches. Use it with care.

All patches are global, so don't *ever* replace an already "shipped" patch in this way! If an erroneous patch is deleted and replaced with a better one, you have to replace it in *all* repositories that have a copy of it. This may not be feasible, unless it's all private repositories. If other developers have already made patches or tags in their repositories that depend on the old patch, things will get complicated.

4.1.4 Special patches and pending

The patches described in the previous sections have mostly been hunks. A *hunk* is one of darcs' primitive patch types, and it is used to remove old lines and/or insert new lines. There are other types of primitive patches, such as *adddir* and *addfile* which add new directories and files, and *replace* which does a search-and-replace on tokens in files.

Hunks are always calculated in place with a diff algorithm just before `whatsnew` or `record`. But other types of primitive patches need to be explicitly created with a darcs command. They are kept in *pending*, ie, in the file `_darcs/patches/pending`, until they are either recorded or reverted.

Pending can be thought of as a special extension of *working*. When you issue, e.g., a darcs `replace` command, the replace is performed on the files in *working* and at the same time a replace patch is put in *pending*. Patches in *pending* describe special changes made in *working*. The diff algorithm will fictively apply these changes to *pristine* before it compares it to *working*, so all lines in *working* that are changed by a `replace` command will also be changed in *pending* and *pristine* when the hunks are calculated. That's why no hunks with the replaced lines will be shown by `whatsnew`; it only shows the replace patch in *pending* responsible for the change.

If a special patch is recorded, it will simply be moved to *pristine*. If it is instead reverted, it will be deleted from *pending* and the accompanying change will be removed from *working*.

Note that reverting a patch in pending is *not* the same as simply removing it from pending. It actually applies the inverse of the change to *working*. Most notable is that reverting an addfile patch will delete the file in *working* (the inverse of adding it). So if you add the wrong file to darcs by mistake, *don't* `revert` the addfile. Instead use `remove`, which cancels out the addfile in pending.

4.2 Using patches

This section will lay down the concepts around patch distribution and branches. The aim is to develop a way of thinking that corresponds well to how darcs is behaving — even in complicated situations.

A repository is a collection of patches. Patches have no defined order, but patches can have dependencies on other patches. Patches can be added to a repository in any order as long as all patches depended upon are there. Patches can be removed from a repository in any order, as long as no remaining patches depend on them.

Repositories can be cloned to create branches. Patches created in different branches may conflict. A conflict is a valid state of a repository. A conflict makes the working tree ambiguous until the conflict is resolved.

4.2.1 Dependencies

There are two kinds of dependencies.

Implicit dependencies is the far most common kind. These are calculated automatically by darcs. If a patch removes a file or a line of code, it will have to depend on the patch that added that file or line of code. If a patch adds a line of code, it will usually have to depend on the patch or patches that added the adjacent lines.

Explicit dependencies can be created if you give the `--ask-deps` option to `darcs record`. This is good for assuring that logical dependencies hold between patches. It can also be used to group patches—a patch with explicit dependencies doesn't need to change anything—and pulling the patch also pulls all patches it was made to depend on.

4.2.2 Branches: just normal repositories

Darcs does not have branches—it doesn't need to. Any two repositories are "branches" in darcs, but it is not of much use unless they have a large portion of patches in common. If they are different projects they will have nothing in common, but darcs may still very well be able to merge them, although the result probably is nonsense. Therefore the word "branch" isn't a technical term in darcs; it's just the way we think of one repository in relation to another.

Branches are *very* useful in darcs. They are in fact *necessary* if you want to do more than only simple work. When you `get` someone's repository from the Internet, you are actually creating a branch of it. But darcs is designed this way, and it has means to make it efficient. The answer to many questions about how to do a thing with darcs is: "use a branch". It is a simple and elegant solution with great power and flexibility, which contributes to darcs' uncomplicated user interface.

You create new branches (i.e., clone repositories) with the `get` and `put` commands.

4.2.3 Moving patches around—no versions

Patches are global, and a copy of a patch either is or is not present in a branch. This way you can rig a branch almost any way you like, as long as dependencies are fulfilled—darcs *won't* let you break dependencies. If you suspect a certain feature from some time ago introduced a bug, you can remove the patch/patches that adds the feature, and try without it.[3]

Patches are added to a repository with `pull` and removed from the repositories with `obliterate`. Don't confuse these two commands with `record` and `unrecord`, which constructs and deconstructs patches.

It is important not to lose patches when (re)moving them around. `pull` needs a source repository to copy the patch from, whereas `obliterate` just erases the patch. Beware that if you obliterate *all* copies of a patch it is completely lost—forever. Therefore you should work with branches when you obliterate patches. The `obliterate` command can wisely be disabled in a dedicated main repository by adding `obliterate disable` to the repository's defaults file.

[3] darcs even has a special command, `trackdown` that automatically removes patches until a specified test no longer fails.

For convenience, there is a `push` command. It works like `pull` but in the other direction. It also differs from `pull` in an important way: it starts a second instance of darcs to apply the patch in the target repository, even if it's on the same computer. It can cause surprises if you have a "wrong" darcs in your PATH.

4.2.4 Tags—versions

While `pull` and `obliterate` can be used to construct different "versions" in a repository, it is often desirable to name specific configurations of patches so they can be identified and retrieved easily later. This is how darcs implements what is usually known as versions. The command for this is `tag`, and it records a tag in the current repository.

A tag is just a patch, but it only contains explicit dependencies. It will depend on all the patches in the current repository.[4] Darcs can recognize if a patch is as a tag; tags are sometimes treated specially by darcs commands.

While traditional revision control systems tag versions in the time line history, darcs lets you tag any configuration of patches at any time, and pass the tags around between branches.

With the option `--tag` to `get` you can easily get a named version in the repository as a new branch.

4.2.5 Conflicts

A conflict *happens* when two conflicting patches meet in the same repository. This is no problem for darcs; it can happily pull together just any patches. But it is a problem for the files in *working* (and *pristine*). The conflict can be thought of as two patches telling darcs different things about what a file should look like.

Darcs escapes this problem by ignoring those parts of the patches that conflict. They are ignored in *both* patches. If patch A changes the line "FIXME" to "FIXED", and patch B changes the same line to "DONE", the two patches together will produce the line "FIXME". Darcs doesn't care which one you pulled into the repository first, you still get the same result when the conflicting patches meet. All other changes made by A and B are performed as normal.

Darcs can mark a conflict for you in *working*. This is done with `mark-conflicts`. Conflicts are marked such that both conflicting changes are inserted with special delimiter lines around them. Then you can merge the two changes by hand, and remove the delimiters.

When you pull patches, darcs automatically performs a `mark-conflicts` for you if a conflict happens. You can remove the markup with `revert`, Remember that the result will be the lines from the previous version common to both conflicting patches. The conflict marking can be redone again with `mark-conflicts`.

A special case is when a pulled patch conflicts with unrecorded changes in the repository. The conflict will be automatically marked as usual, but since the markup is *also* an unrecorded change, it will get mixed in with your unrecorded changes. There is no guarantee you can revert *only* the markup after this, and `mark-conflicts` will not be able to redo this markup later if you remove it. It is good practice to record important changes before pulling.

`mark-conflicts` can't mark complicated conflicts. In that case you'll have to use `darcs diff` and other commands to understand what the conflict is all about. If for example two conflicting patches create the same file, `mark-conflicts` will pick just one of them, and no delimiters are inserted. So watch out if darcs tells you about a conflict.

`mark-conflicts` can also be used to check for unresolved conflicts. If there are none, darcs replies "No conflicts to mark.". While `pull` reports when a conflict happens, `obliterate` and `get` don't.

4.2.6 Resolving conflicts

A conflict is resolved (not marked, as with the command `mark-conflicts`) as soon as some new patch depends on the conflicting patches. This will usually be the resolve patch you record after manually putting together the pieces from the conflict markup produced by `mark-conflicts` (or `pull`). But it can just as well be a tag. So don't forget to fix conflicts before you accidentally "resolve" them by recording other patches.

[4] It will omit patches already depended upon by other patches, since they will be indirectly depended upon anyway.

If the conflict is with one of your not-yet-published patches, you may choose to amend that patch rather than creating a resolve patch.

If you want to back out and wait with the conflict, you can `obliterate` the conflicting patch you just pulled. Before you can do that you have to `revert` the conflict markups that `pull` inserted when the conflict happened.

4.3 Global and per-repository caches

Darcs uses a global cache, as this is one of its biggest performance enhancing tools. The global cache acts as a giant patch pool where darcs first looks for a patch when grabbing new patches. This saves time by not downloading the same patch twice from a remote server. It also saves space by storing the patch only once, if you ensure your cache and your repositories are on the same hardlink-supporting filesystem.

Darcs now enables a global patch cache under your home directory by default. Older darcs 2.x versions required this manual step:

```
$ mkdir -p $HOME/.darcs/cache
$ echo cache:$HOME/.darcs/cache > $HOME/.darcs/sources
```

On MS Windows 3, using `cmd.exe` (Command Prompt under Accessories):

```
> md "%APPDATA%\darcs\cache" (notice double quotes!)
> echo cache:%APPDATA%\darcs\cache > "%APPDATA%\darcs\sources"
```

Individual repositories also contain cache location information. Each time a repository is got, its location is added as an entry in `_darcs/prefs/sources`. If one of these repositories were to become totally or temporarily unreachable, it can cause darcs to hang for a long time trying to reach it. Fortunately darcs has a mechanism which helps us to deal with that problem: if an unreachable entry is discovered, darcs stops using it for the rest of the session and then notifies the user to take further action. It will display a message like the following:

```
> I could not reach the following repository:
> http://darcs.net/
> If you're not using it, you should probably delete
> the corresponding entry from _darcs/prefs/sources.
```

There are some other advanced things you can do in `_darcs/prefs/sources`, such as create read-only caches and even set a primary source repository above any used in a `darcs get` or `darcs pull` command.

4.4 Distributed development with one primary developer

This is how a project with many contributors, but every contribution is reviewed and manually applied by the project leader, can be run. For this sort of a situation, `darcs send` is ideal, since the barrier for contributions is very low, which helps encourage contributors.

One could simply set the `_darcs/prefs/email` value to the project mailing list, but you may also want to use darcs send to send your changes to the main server, so instead the email address could be set to something like "`Darcs Repo <maintainer@host.com>`". Then the `.procmailrc` file on the server should have the following rule:

```
:0
* ^TODarcs Repo
|(umask 022; darcs apply --reply project-mailing-list@host.com \
             --repodir /path/to/repo --verify /path/to/allowed_keys)
```

This causes darcs apply to be run on any email sent to "Darcs Repo". `apply` actually applies them only if they are signed by an authorized key.

The central darcs repository contains the following values in its `_darcs/prefs/defaults`:

```
apply test
apply verbose
apply happy-forwarding
```

The first line tells apply to always run the test suite. The test suite can be a reason to use send rather than push, since it allows to easily continue working (or put one's computer to sleep) while the tests are being run on the main server. The second line is just there to improve the email response that the maintainer gets when a patch has either been applied or failed the tests. The third line makes darcs not complain about unsigned patches, but just to forward them to `darcs-devel`.

On one's development computer, one can have in their `.muttrc` the following alias, which allows to easily apply patches received via email directly to one's darcs working directory:

```
macro pager A "<pipe-entry>(umask 022; darcs apply --no-test -v \
       --repodir ~/darcs)"
```

4.5 Sending signed patches by email

Darcs send can be configured to send a cryptographically signed patch by email. You can then set up your mail system to have darcs verify that patches were signed by an authorized user and apply them when a patch is received by email. The results of the apply can be returned to the user by email. Unsigned patches (or patches signed by unauthorized users) will be forwarded to the repository owner (or whoever you configure them to be forwarded to...).

This method is especially nice when combined with the `--test` option of darcs apply, since it allows you to run the test suite (assuming you have one) and reject patches that fail—and it's all done on the server, so you can happily go on working on your development machine without slowdown while the server runs the tests.

Setting up darcs to run automatically in response to email is by far the most complicated way to get patches from one repository to another... so it'll take a few sections to explain how to go about it.

Security considerations When you set up darcs to run apply on signed patches, you should assume that a user with write access can write to any file or directory that is writable by the user under which the apply process runs. Unless you specify the `--no-test` flag to darcs apply (and this is *not* the default), you are also allowing anyone with write access to that repository to run arbitrary code on your machine (since they can run a test suite—which they can modify however they like). This is quite a potential security hole.

For these reasons, if you don't implicitly trust your users, it is recommended that you create a user for each repository to limit the damage an attacker can do with access to your repository. When considering who to trust, keep in mind that a security breach on any developer's machine could give an attacker access to their private key and passphrase, and thus to your repository.

Installing necessary programs You also must install the following programs: gnupg, a mailer configured to receive mail (e.g. exim, sendmail or postfix), and a web server (usually apache).

Granting access to a repository You create your gpg key by running (as your normal user):

```
$ gpg --gen-key
```

You will be prompted for your name and email address, among other options. Of course, you can skip this step if you already have a gpg key you wish to use.

You now need to export the public key so we can tell the patcher about it. You can do this with the following command (again as your normal user):

```
$ gpg --export "email@address" > /tmp/exported_key
```

And now we can add your key to the `allowed_keys`:

```
(as root)> gpg --keyring /var/lib/darcs/repos/myproject/allowed_keys \
               --no-default-keyring --import /tmp/exported_key
```

You can repeat this process any number of times to authorize multiple users to send patches to the repository.

You should now be able to send a patch to the repository by running as your normal user, in a working copy of the repository:

```
$ darcs send --sign http://your.computer/repos/myproject
```

You may want to add "send sign" to the file `_darcs/prefs/defaults` so that you won't need to type `--sign` every time you want to send...

If your gpg key is protected by a passphrase, then executing `send` with the `--sign` option might give you the following error:

```
darcs failed:  Error running external program 'gpg'
```

The most likely cause of this error is that you have a misconfigured gpg that tries to automatically use a non-existent gpg-agent program. GnuPG will still work without gpg-agent when you try to sign or encrypt your data with a passphrase protected key. However, it will exit with an error code 2 (`ENOENT`) causing `darcs` to fail. To fix this, you will need to edit your `~/.gnupg/gpg.conf` file and comment out or remove the line that says:

```
use-agent
```

If after commenting out or removing the `use-agent` line in your gpg configuration file you still get the same error, then you probably have a modified GnuPG with use-agent as a hard-coded option. In that case, you should change `use-agent` to `no-use-agent` to disable it explicitly.

Setting up a sendable repository using procmail If you don't have root access on your machine, or perhaps simply don't want to bother creating a separate user, you can set up a darcs repository using procmail to filter your mail. Let us assume that you already use procmail to filter your email. If not, you will need to read up on it, or perhaps should use a different method for routing the email to darcs.

To begin with, you must configure your repository so that a darcs send to your repository will know where to send the email. Do this by creating a file in `/path/to/your/repo/_darcs/prefs` called `email` containing your email address. As a trick (to be explained below), we will create the email address with "darcs repo" as your name, in an email address of the form "User Name <user@host.com>."

```
$ echo 'my darcs repo <user@host.com>' \
      > /path/to/your/repo/_darcs/prefs/email
```

The next step is to set up a gnupg keyring containing the public keys of people authorized to send to your repository. Here is a second way of going about this (see above for the first). This time let us assume you want to give someone write access to your repository. You can do this by:

```
gpg --no-default-keyring \
    --keyring /path/to/the/allowed_keys --recv-keys FFFFFFFF
```

With "FFFFFFFF" being the ID of the gpg key of the person to be authorised, if this person has uploaded their key to the gpg keyservers. Actually, this also requires that you have configured gpg to access a valid keyserver. You can, of course, repeat this command for all keys you want to allow access to.

Finally, we add a few lines to your `.procmailrc`:

```
:0
* ^TOmy darcs repo
|(umask 022; darcs apply --reply user@host.com \
    --repodir /path/to/your/repo --verify /path/to/the/allowed_keys)
```

The purpose for the "my darcs repo" trick is partially to make it easier to recognize patches sent to the repository, but is even more crucial to avoid nasty bounce loops by making the `--reply` option have an email address that won't go back to the repository. This means that unsigned patches that are sent to your repository will be forwarded to your ordinary email.

Like most mail-processing programs, Procmail by default sets a tight umask. However, this will prevent the repository from remaining world-readable; thus, the "umask 022" is required to relax the umask. (Alternatively, you could set Procmail's global `UMASK` variable to a more suitable value.)

Checking if your e-mail patch was applied After sending a patch with `darcs send`, you may not receive any feedback, even if the patch is applied. You can confirm whether or not your patch was applied to the remote repository by pointing `darcs changes` at a remote repository:

```
darcs changes --last=10 --repo=http://darcs.net/
```

That shows you the last 10 changes in the remote repository. You can adjust the options given to `changes` if a more advanced query is needed.

Chapter 5

Darcs commands

The general format of a darcs command is

```
% darcs COMMAND OPTIONS ARGUMENTS ...
```

Here `COMMAND` is a command such as `add` or `record`, which of course may have one or more arguments. Options have the form `--option` or `-o`, while arguments vary from command to command. There are many options which are common to a number of different commands, which will be summarized here.

If you wish, you may use any unambiguous beginning of a command name as a shortcut: for `darcs record`, you could type `darcs recor` or `darcs rec`, but not `darcs re` since that could be confused with `darcs replace`, `darcs revert` and `darcs remove`.

In some cases, `COMMAND` actually consists of two words, a super-command and a subcommand. For example, the "display the manifest" command has the form `darcs query manifest`.

Command overview Not all commands modify the "patches" of your repository (that is, the named patches which other users can pull); some commands only affect the copy of the source tree you're working on (your "working directory"), and some affect both. This table summarizes what you should expect from each one and will hopefully serve as guide when you're having doubts about which command to use.

affects	patches	working directory
record	yes	no
unrecord	yes	no
rollback	yes	yes
revert	no	yes
unrevert	no	yes
pull	yes	yes
obliterate	yes	yes
apply	yes	yes
push[1]	no	no
send[2]	no	no
put[3]	no	no

5.1 Common options to darcs commands

`--help`

[1] But it affects the repository and working directory targeted by the push

[2] As for the other end, see apply

Every `COMMAND` accepts `--help` as an argument, which tells it to provide a bit of help. Among other things, this help always provides an accurate listing of the options available with that command, and is guaranteed never to be out of sync with the version of darcs you actually have installed (unlike this manual, which could be for an entirely different version of darcs).

```
% darcs COMMAND --help
```

`--disable`

Every `COMMAND` accepts the `--disable` option, which can be used in `_darcs/prefs/defaults` to disable some commands in the repository. This can be helpful if you want to protect the repository from accidental use of advanced commands like obliterate, unpull, unrecord or amend-record.

`--verbose, --quiet, --normal-verbosity`

Most commands also accept the `--verbose` option, which tells darcs to provide additional output. The amount of verbosity varies from command to command. Commands that accept `--verbose` also accept `--quiet`, which surpresses non-error output, and `--normal-verbosity` which can be used to restore the default verbosity if `--verbose` or `--quiet` is in the defaults file.

`--debug, --debug-http`

Many commands also accept the `--debug` option, which causes darcs to generate additional output that may be useful for debugging its behavior, but which otherwise would not be interesting. Option `--debug-http` makes darcs output debugging info for libcurl.

`--repodir`

Another common option is the `--repodir` option, which allows you to specify the directory of the repository in which to perform the command. This option is used with commands, such as whatsnew, that ordinarily would be performed within a repository directory, and allows you to use those commands without actually being in the repository directory when calling the command. This is useful when running darcs in a pipe, as might be the case when running `apply` from a mailer.

`--remote-repo`

Some commands, such as `pull` require a remote repository to be specified, either from the command line or as a default. The `--remote-repo` provides an alternative way to supply this remote repository path. This flag can be seen as temporarily "replacing" the default repository. Setting it causes the command to ignore the default repository (it also does not affect, i.e. overwrite the default repository). On the other hand, if any other repositories are supplied as command line arguments, this flag will be ignored (and the default repository may be overwritten).

Selecting patches Many commands operate on a patch or patches that have already been recorded. There are a number of options that specify which patches are selected for these operations: `--patch`, `--match`, `--tag`, and variants on these, which for `--patch` are `--patches`, `--from-patch`, and `--to-patch`. The `--patch` and `--tag` forms simply take (POSIX extended, aka `egrep`) regular expressions and match them against tag and patch names. `--match`, described below, allows more powerful patterns.

The plural forms of these options select all matching patches. The singular forms select the last matching patch. The range (from and to) forms select patches after or up to (both inclusive) the last matching patch.

These options use the current order of patches in the repository. darcs may reorder patches, so this is not necessarily the order of creation or the order in which patches were applied. However, as long as you are just recording patches in your own repository, they will remain in order.

When a patch or a group of patches is selected, all patches they depend on get silently selected too. For example: `darcs pull --patches bugfix` means "pull all the patches with 'bugfix' in their name, along with any patches they require." If you really only want patches with 'bugfix' in their name, you should use the `--no-deps` option, which makes darcs exclude any matched patches from the selection which have dependencies that are themselves not explicitly matched by the selection.

For `unrecord`, `unpull` and `obliterate`, patches that depend on the selected patches are silently included, or if `--no-deps` is used selected patches with dependencies on not selected patches are excluded from the selection.

5.1.1 Match

Currently `--match` accepts eight primitive match types, although there are plans to expand it to match more patterns. Also, note that the syntax is still preliminary and subject to change.

The first match type accepts a literal string which is checked against the patch name. The syntax is

```
darcs annotate --summary --match 'exact foo+bar'
```

This is useful for situations where a patch name contains characters that could be considered special for regular expressions.

In this and the other match types, the argument must be enclosed in double quotes if it contains spaces. You can escape a quote in the argument with a backslash; backslash escapes itself, but it is treated literally if followed by a character other than a double quote or backslash, so it is typically not necessary to escape a backslash. No such escaping is necessary unless the argument is enclosed in double quotes.

The second match type accepts a regular expression which is checked against the patch name. The syntax is

```
darcs annotate --summary --match 'name foo'
```

Note that to match regexp metacharacters, such as `(`, literally, they must be escaped with backslash along with any embedded double quotes. To match a literal backslash it must be written quadrupled in general, but often it need not be escaped, since backslash is only special in regexps when followed by a metacharacter. In the following example pairs, the first literal is matched by the second sequence in the match name: "`"`":"`\"`", "`\`":"`\\\\`", "`\x`":"`\x`", "`(`":"`\(`".

The third match type matches the darcs hash for each patch:

```
darcs annotate --summary --match \
  'hash 20040403105958-53a90-c719567e92c3b0ab9eddd5290b705712b8b918ef'
```

Note you need to provide the full hash string as above. This is intended to be used, for example, by programs allowing you to view darcs repositories (e.g. CGI scripts like viewCVS).

The fourth match type accepts a regular expression which is checked against the patch author. The syntax is

```
darcs annotate --summary --match 'author foo'
```

There is also support for matching by date. This is done using commands such as

```
darcs annotate --summary --match 'date "last week"'
darcs annotate --summary --match 'date yesterday'
darcs annotate --summary --match 'date "today 14:00"'
```

```
darcs annotate --summary --match 'date "tea time yesterday"'
darcs annotate --summary --match 'date "3 days before last year at 17:00"'
darcs changes --from-match 'date "Sat Jun  30 11:31:30 EDT 2004"'
```

Only English date specifications are supported—specifically you must use English day and month names. Also, only a limited set of time zones is supported (compatible with GNU coreutils' date parsing). Unrecognized zones are treated as UTC, which may result in the timestamps printed in change entries not being recognized by the date matching. You can avoid this problem on a POSIX-like system by running darcs in the UTC zone to get the times initially, e.g.:

```
TZ=UTC darcs changes
```

When matching on the ISO format, a partial date is treated as a range. English dates can either refer to a specific day ("6 months ago','day before yesterday"), or to an interval from some past date ("last month") to the present. Putting this all together, if today is "2004-07-24" then the following matches should work:

date	**patches selected**
2004	from 2004-01-01 up to and including 2004-12-31
2004-01	from 2004-01-01 up to and including 2004-01-31
2004-01-01	during 2004-01-01
today	during 2004-07-24 (starting midnight in your timezone)
yesterday	during 2004-07-23
6 months ago	during 2004-01-23
last 6 months	since 2004-01-23
last month	since 2004-06-23 (not 2004-06-01!)
last week	since 2004-07-16

For more precise control, you may specify an interval, either in a small subset of English or of the ISO 8601 format[4]. If you use the ISO format, note that durations, when specified alone, are interpreted as being relative to the current date and time.

```
darcs annotate --summary --match 'date "between 2004-03-12 and last week"'
darcs annotate --summary --match 'date "after 2005"'
darcs annotate --summary --match 'date "in the last 3 weeks"'
darcs annotate --summary --match 'date "P3M/2006-03-17"'
darcs annotate --summary --match 'date "2004-01-02/2006-03-17"'
darcs annotate --summary --match 'date "P2M6D"'
```

You may also prefer to combine date matching with a more specific pattern.

```
darcs annotate --summary --match 'date "last week" && name foo'
```

The sixth match type accepts a regular expression which is checked against file paths that the patch touches. The syntax is

```
darcs annotate --summary --match 'touch foo/bar.c'
```

The seventh match type accepts a regular expression which is checked against every hunk. The syntax is

```
darcs annotate --summary --match 'hunk "^instance .* Foo where$"'
```

The eight match type accepts a regular expression which is checked against the long comment. The syntax is

```
darcs annotate --summary --match 'comment "remote repository"'
```

The `--match` pattern can include the logical operators `&&`, `||` and `not`, as well as grouping of patterns with parentheses. For example

[4] http://www.w3.org/TR/NOTE-datetime

```
darcs annotate --summary --match 'name record && not name overrode'
```

`--ignore-times, --no-ignore-times`

Darcs optimizes its operations by keeping track of the modification times of your files. This dramatically speeds up commands such as `whatsnew` and `record` which would otherwise require reading every file in the repository and comparing it with a reference version. However, there are times when this can cause problems, such as when running a series of darcs commands from a script, in which case often a file will be modified twice in the same second, which can lead to the second modification going unnoticed. The solution to such predicaments is the `--ignore-times` option, which instructs darcs not to trust the file modification times, but instead to check each file's contents explicitly.

`--author`

DARCS_EMAIL and EMAIL Each patch is attributed to its author, usually by email address (for example, 'Fred Bloggs ¡fred@example.net¿'). Darcs looks in several places for this author string: the –author option, the files _darcs/prefs/author (in the repository) and /.darcs/author (in your home directory), and the environment variables $DARCS_EMAIL and $EMAIL. If none of those exist, Darcs will prompt you for an author string and write it to _darcs/prefs/author. Note that if if you have more than one email address, note that you can put them all in /.darcs/author, one author per line. Darcs will still prompt you for an author, but it allows you to select from the list, or to type in an alternative.

`--dont-compress, --compress`

By default, darcs commands that write patches to disk will compress the patch files. If you don't want this, you can choose the `--dont-compress` option, which causes darcs not to compress the patch file.

`--dry-run`

The `--dry-run` option will cause darcs not to actually take the specified action, but only print what would have happened. Not all commands accept `--dry-run`, but those that do should accept the `--summary` option.

`--summary, --no-summary`

The `--summary` option shows a summary of the patches that would have been pulled/pushed/whatever. The format is similar to the output format of `cvs update` and looks like this:

```
A  ./added_but_not_recorded.c
A! ./added_but_not_recorded_conflicts.c
a  ./would_be_added_if_look_for_adds_option_was_used.h

M  ./modified.t -1 +1
M! ./modified_conflicts.t -1 +1

R  ./removed_but_not_recorded.c
R! ./removed_but_not_recorded_conflicts.c
```

You can probably guess what the flags mean from the clever file names.

`A` is for files that have been added but not recorded yet.

`a` is for files found using the `--look-for-adds` option available for `whatsnew` and `record`. They have not been added yet, but would be added automatically if `--look-for-adds` were used with the next `record` command.

`M` is for files that have been modified in the working directory but not recorded yet. The number of added and subtracted lines is also shown.

`R` is for files that have been removed, but the removal is not recorded yet.

An exclamation mark appears next to any option that has a conflict.

Resolution of conflicts To resolve conflicts using an external tool, you need to specify a command to use, e.g.

```
--external-merge 'opendiff %1 %2 -ancestor %a -merge %o'
```

The `%1` and `%2` are replaced with the two versions to be merged, `%a` is replaced with the common ancestor of the two versions. Most importantly, `%o` is replaced with the name of the output file that darcs will require to be created holding the merged version. The above example works with the FileMerge.app tool that comes with Apple's developer tools. To use xxdiff, you would use

```
--external-merge 'xxdiff -m -O -M %o %1 %a %2'
```

To use `kdiff3`, you can use

```
--external-merge 'kdiff3 --output %o %a %1 %2'
```

To use `tortoiseMerge`, you can use

```
--external-merge 'tortoiseMerge /base:"%a" /mine:"%1" /theirs:"%2" /merged:"%o"'
```

(`tortoiseMerge` is a nice merge tool that comes with TortoiseSVN and works well on Windows.)

Note that the command is split into space-separated words and the first one is `exec`ed with the rest as arguments—it is not a shell command. In particular, on Windows this means that the first command path should not contain spaces and you should make sure the command is in your `PATH`.

The substitution of the `%` escapes is done everywhere. If you need to prevent substitution you can use a double percentage sign, i.e. `%%a` is substituted with `%a`. Here is an example script to use the Emacs' Ediff package for merging.

```
#! /bin/sh
# External merge command for darcs, using Emacs Ediff, via server if possible.
# It needs args %1 %2 %a %o, i.e. the external merge command is, say,
# 'emerge3 %1 %2 %a %o'.
test $# -eq 4 || exit 1
form="(ediff-merge-files-with-ancestor"
while test $# -gt 0; do
    count=$count.
    if [ $count = .... ]; then
        form=$form\ nil         # Lisp STARTUP-HOOKS arg
    fi
    case $1 in                  # Worry about quoting -- escape " and \
        *[\"\\]* ) form=$form\ \"$(echo $1 | sed -e's/["\\]/\\\0/g')\" ;;
        *) form=$form\ \"$1\" ;;
```

```
    esac
    shift
done
form=$form')'
( emacsclient --eval "$form" || # Emacs 22 server
  gnudoit "$form" ||            # XEmacs/Emacs 21 server
  emacs --eval "$form" ||       # Relatively slow to start up
  xemacs -eval "$form"          # Horribly slow to start up
) 2>/dev/null
```

It would be invoked like:

```
--external-merge 'emerge3 %1 %2 %a %o'
```

Note that if you do use an external merge tool, most likely you will want to add to your defaults file (`_darcs/prefs/defaults` or `~/.darcs/prefs`, see 3.1, on MS Windows 3) a line such as

```
ALL external-merge kdiff3 --output %o %a %1 %2
```

or

```
ALL external-merge tortoiseMerge /base:"%a" /mine:"%1" /theirs:"%2" /merged:"%o"
```

Note that the defaults file does not want quotes around the command.

`--sendmail-command`

SENDMAIL On Unix, the 'darcs send' command relies on sendmail(8). The '–sendmail-command' or $SENDMAIL environment variable can be used to provide an explicit path to this program; otherwise the standard locations /usr/sbin/sendmail and /usr/lib/sendmail will be tried.

5.1.2 Posthooks

`--posthook=COMMAND, --no-posthook`

To provide a command that should be run whenever a darcs command completes successfully, use `--posthook` to specify the command. This is useful for people who want to have a command run whenever a patch is applied. Using `--no-posthook` will disable running the command.

`--run-posthook, --prompt-posthook`

These options control prompting before running the posthook. Use `--prompt-posthook` to have darcs prompt before running the posthook command. You may use –run-posthook to reenable the default behavior of running user-specified posthooks.

Some darcs commands export to the posthook command information about the changes being made. In particular, three environment variables are defined. `DARCS_PATCHES` contains a human-readable summary of the patches being acted upon. The format is the same as "darcs changes". `DARCS_PATCHES_XML` Contains the same details, in the same XML format as "darcs changes". Finally, `DARCS_FILES` contains a list of the files affected, one file per line. If your repository has filenames including newlines, you'll just have to cope. Note, however, that *none* of these environment variables are defined when running under windows. Note also that we refuse to pass environment variables greater in size than 10k, in order to avoid triggering `E2BIG` errors.

5.1.3 Prehooks

`--prehook=COMMAND, --no-prehook`

To provide a command that should be run before a darcs command is executed, use `--prehook` to specify the command. An example use is for people who want to have a command run whenever a patch is to be recorded, such as translating line endings before recording patches. Using `--no-prehook` will disable running the command.

`--run-prehook, --prompt-prehook`

These options control prompting before running the prehook. See the posthook documentation above for details.

`--http-pipelining, --no-http-pipelining`

When compiled with libcurl (version 7.18.0 and above), darcs can use HTTP pipelining. It is enabled by default for libcurl (version 7.19.1 and above). This option will make darcs enable or disable HTTP pipelining, overwriting default. Note that if HTTP pipelining is really used depends on the server.

`--no-cache`

Do not use patch caches.

`--umask`

By default, Darcs will use your current umask. The option `--umask` will cause Darcs to switch to a different umask before writing to the repository.

`--dont-restrict-paths, --restrict-paths`

By default darcs is only allowed to manage and modify files and directories contained inside the current repository and not being part of any darcs repository's meta data (including the current one). This is mainly for security, to protect you from spoofed patches modifying arbitrary files with sensitive data—say, in your home directory—or tampering with any repository's meta data to switch off this safety guard.

But sometimes you may want to manage a group of "sub" repositories' preference files with a global repository, or use darcs in some other advanced way. The best way is probably to put `ALL dont-restrict-paths` in `_darcs/prefs/defaults`. This turns off all sanity checking for file paths in patches.

Path checking can be temporarily turned on with `--restrict-paths` on the command line, when pulling or applying unknown patches.

`--allow-unrelated-repos`

By default darcs checks and warns user if repositories are unrelated when doing pull, push and send. This option makes darcs skip this check.

5.2 Options apart from darcs commands

`--help`

Calling darcs with just `--help` as an argument gives a brief summary of what commands are available.

`--version, --exact-version`

Calling darcs with the flag `--version` tells you the version of darcs you are using. Calling darcs with the flag `--exact-version` gives the precise version of darcs, even if that version doesn't correspond to a released version number. This is helpful with bug reports, especially when running with a "latest" version of darcs.

`--commands`

Similarly calling darcs with only `--commands` gives a simple list of available commands. This latter arrangement is primarily intended for the use of command-line autocompletion facilities, as are available in bash.

5.3 Getting help

5.3.1 darcs help

Without arguments, 'darcs help' prints a categorized list of darcs commands and a short description of each one. With an extra argument, 'darcs help foo' prints detailed help about the darcs command foo.

`Usage: darcs help [OPTION]... [<DARCS_COMMAND> [DARCS_SUBCOMMAND]]`

Options:

Display help about darcs and darcs commands.

5.4 Creating repositories

5.4.1 darcs initialize

The 'darcs initialize' command turns the current directory into a Darcs repository. Any existing files and subdirectories become UNSAVED changes: record them with 'darcs record –look-for-adds'.

This command creates the '_darcs' directory, which stores version control metadata. It also contains per-repository settings in _darcs/prefs/, which you can read about in the user manual.

By default, patches of the new repository are in the darcs-2 semantics. However it is possible to create a repository in darcs-1 semantics with the flag '–hashed', althought this is not recommended except for sharing patches with a project that uses patches in the darcs-1 semantics.

Initialize is commonly abbreviated to 'init'.

`Usage: darcs initialize [OPTION]...`

Options:

`--hashed`	Some new features. Compatible with older repos
`--darcs-2`	All features. Related repos must use same format [DEFAULT].
`--with-working-dir`	Create a working directory (normal repository)
`--no-working-dir`	Do not create a working directory (bare repository)
`--repodir DIRECTORY`	specify the repository directory in which to run

Make the current directory a repository.

5.4.2 darcs get

Get creates a local copy of a repository. The optional second argument specifies a destination directory for the new copy; if omitted, it is inferred from the source location.

By default Darcs will copy every patch from the original repository. This means the copy is completely independent of the original; you can operate on the new repository even when the original is inaccessible. If you expect the original repository to remain accessible, you can use –lazy to avoid copying patches until they are needed ('copy on demand'). This is particularly useful when copying a remote repository with a long history that you don't care about.

The –lazy option isn't as useful for local copies, because Darcs will automatically use 'hard linking' where possible. As well as saving time and space, you can move or delete the original repository without affecting a complete, hard-linked copy. Hard linking requires that the copy be on the same filesystem and the original repository, and that the filesystem support hard linking. This includes NTFS, HFS+ and all general-purpose Unix filesystems (such as ext3, UFS and ZFS). FAT does not support hard links.

Darcs get will not copy unrecorded changes to the source repository's working tree.

It is often desirable to make a copy of a repository that excludes some patches. For example, if releases are tagged then 'darcs get –tag .' would make a copy of the repository as at the latest release.

An untagged repository state can still be identified unambiguously by a context file, as generated by 'darcs changes –context'. Given the name of such a file, the –context option will create a repository that includes only the patches from that context. When a user reports a bug in an unreleased version of your project, the recommended way to find out exactly what version they were running is to have them include a context file in the bug report.

You can also make a copy of an untagged state using the –to-patch or –to-match options, which exclude patches 'after' the first matching patch. Because these options treat the set of patches as an ordered sequence, you may get different results after reordering with 'darcs optimize', so tagging is preferred.

```
Usage: darcs get [OPTION]... <REPOSITORY> [<DIRECTORY>]
```

Options:

	`--repo-name DIRECTORY,--repodir DIRECTORY`	path of output directory
	`--lazy`	get patch files only as needed
	`--complete`	get a complete copy of the repository
	`--to-match PATTERN`	select changes up to a patch matching PATTERN
	`--to-patch REGEXP`	select changes up to a patch matching REGEXP
`-t`	`--tag REGEXP`	select tag matching REGEXP
	`--context FILENAME`	version specified by the context in FILENAME
	`--set-default`	set default repository [DEFAULT]
	`--no-set-default`	don't set default repository
	`--set-scripts-executable`	make scripts executable
	`--dont-set-scripts-executable,--no-set-scripts-executable`	don't make scripts executable
	`--with-working-dir`	Create a working directory (normal repository)
	`--no-working-dir`	Do not create a working directory (bare repository)

Advanced options:

`--packs`	use repository packs [DEFAULT]
`--no-packs`	don't use repository packs
`--no-http-pipelining`	disable HTTP pipelining
`--remote-darcs COMMAND`	name of the darcs executable on the remote server

Create a local copy of a repository.

5.5 Modifying the contents of a repository

5.5.1 darcs add

Generally a repository contains both files that should be version controlled (such as source code) and files that Darcs should ignore (such as executables compiled from the source code). The 'darcs add' command is used to tell Darcs which files to version control.

When an existing project is first imported into a Darcs repository, it is common to run 'darcs add -r *' or 'darcs record -l' to add all initial source files into darcs.

Adding symbolic links (symlinks) is not supported.

Darcs will ignore all files and folders that look 'boring'. The –boring option overrides this behaviour.

Darcs will not add file if another file in the same folder has the same name, except for case. The –case-ok option overrides this behaviour. Windows and OS X usually use filesystems that do not allow files a folder to have the same name except for case (for example, 'ReadMe' and 'README'). If –case-ok is used, the repository might be unusable on those systems!

```
Usage: darcs add [OPTION]... <FILE or DIRECTORY> ...
```

Options:

	`--boring`	don't skip boring files
	`--no-boring`	skip boring files [DEFAULT]
	`--case-ok`	don't refuse to add files differing only in case
	`--no-case-ok`	refuse to add files whose name differ only in case [DEFAULT]
	`--reserved-ok`	don't refuse to add files with Windows-reserved names
	`--no-reserved-ok`	refuse to add files with Windows-reserved names [DEFAULT]
`-r`	`--recursive`	add contents of subdirectories
	`--not-recursive,--no-recursive`	don't add contents of subdirectories
	`--repodir DIRECTORY`	specify the repository directory in which to run
	`--dry-run`	don't actually take the action

Advanced options:

`--umask UMASK`	specify umask to use when writing

Add one or more new files or directories.

5.5.2 darcs remove

The 'darcs remove' command exists primarily for symmetry with 'darcs add', as the normal way to remove a file from version control is simply to delete it from the working tree. This command is only useful in the unusual case where one wants to record a removal patch WITHOUT deleting the copy in the working tree (which can be re-added).

Note that applying a removal patch to a repository (e.g. by pulling the patch) will ALWAYS affect the working tree of that repository.

```
Usage: darcs remove [OPTION]... <FILE or DIRECTORY> ...
```

Options:

	`--repodir DIRECTORY`	specify the repository directory in which to run
`-r`	`--recursive`	recurse into subdirectories
	`--not-recursive,--no-recursive`	don't recurse into subdirectories

Advanced options:

`--umask UMASK`	specify umask to use when writing

Remove files from version control.

5.5.3 darcs move

Darcs cannot reliably distinguish between a file being deleted and a new one added, and a file being moved. Therefore Darcs always assumes the former, and provides the 'darcs mv' command to let Darcs know when you want the latter. This command will also move the file in the working tree (unlike 'darcs remove'), unless it has already been moved.

Darcs will not rename a file if another file in the same folder has the same name, except for case. The –case-ok option overrides this behaviour. Windows and OS X usually use filesystems that do not allow files a folder to have the same name except for case (for example, 'ReadMe' and 'README'). If –case-ok is used, the repository might be unusable on those systems!

```
Usage: darcs move [OPTION]... <SOURCE> ... <DESTINATION>
```

Options:

`--case-ok`	don't refuse to add files differing only in case
`--no-case-ok`	refuse to add files whose name differ only in case [DEFAULT]
`--reserved-ok`	don't refuse to add files with Windows-reserved names
`--no-reserved-ok`	refuse to add files with Windows-reserved names [DEFAULT]
`--repodir DIRECTORY`	specify the repository directory in which to run

Advanced options:

`--umask UMASK`	specify umask to use when writing

Move or rename files.

5.5.4 darcs replace

In addition to line-based patches, Darcs supports a limited form of lexical substitution. Files are treated as sequences of words, and each occurrence of the old word is replaced by the new word. This is intended to provide a clean way to rename a function or variable. Such renamings typically affect lines all through the source code, so a traditional line-based patch would be very likely to conflict with other branches, requiring manual merging.

Files are tokenized according to one simple rule: words are strings of valid token characters, and everything between them (punctuation and whitespace) is discarded. By default, valid token characters are letters, numbers and the underscore (i.e. [A-Za-z0-9_]). However if the old and/or new token contains either a hyphen or period, BOTH hyphen and period are treated as valid (i.e. [A-Za-z0-9_.-]).

The set of valid characters can be customized using the –token-chars option. The argument must be surrounded by square brackets. If a hyphen occurs between two characters in the set, it is treated as a set range. For example, in most locales [A-Z] denotes all uppercase letters. If the first character is a caret, valid tokens are taken to be the complement of the remaining characters. For example, [^:\n] could be used to match fields in the passwd(5), where records and fields are separated by newlines and colons respectively.

If you choose to use –token-chars, you are STRONGLY encouraged to do so consistently. The consequences of using multiple replace patches with different –token-chars arguments on the same file are not well tested nor well understood.

By default Darcs will refuse to perform a replacement if the new token is already in use, because the replacements would be not be distinguishable from the existing tokens. This behaviour can be overridden by supplying the –force option, but an attempt to 'darcs rollback' the resulting patch will affect these existing tokens.

Limitations:

The tokenizer treats files as byte strings, so it is not possible for –token-chars to include multi-byte characters, such as the non-ASCII parts of UTF-8. Similarly, trying to replace a 'high-bit' character from a unibyte encoding will also result in replacement of the same byte in files with different encodings. For example, an acute a from ISO 8859-1 will also match an alpha from ISO 8859-7.

Due to limitations in the patch file format, –token-chars arguments cannot contain literal whitespace. For example, [^ \n\t] cannot be used to declare all characters except the space, tab and newline as valid within a word, because it contains a literal space.

Unlike POSIX regex(7) bracket expressions, character classes (such as [[:alnum:]]) are NOT supported by –token-chars, and will be silently treated as a simple set of characters.

```
Usage: darcs replace [OPTION]... <OLD> <NEW> <FILE> ...
```

Options:

```
     --token-chars "[CHARS]"  define token to contain these characters
 -f  --force                  proceed with replace even if 'new' token
                              already exists
     --no-force               don't force the replace if it looks scary
     --repodir DIRECTORY      specify the repository directory in which to
                              run
```

Advanced options:

```
    --ignore-times      don't trust the file modification times
    --no-ignore-times   trust modification times to find modified files
                        [DEFAULT]
    --umask UMASK       specify umask to use when writing
```

Substitute one word for another.

5.6 Working with changes

5.6.1 darcs record

The 'darcs record' command is used to create a patch from changes in the working tree. If you specify a set of files and directories, changes to other files will be skipped.

Every patch has a name, an optional description, an author and a date.

The patch name should be a short sentence that concisely describes the patch, such as 'Add error handling to main event loop.' You can supply it in advance with the -m option, or provide it when prompted.

The patch description is an optional block of free-form text. It is used to supply additional information that doesn't fit in the patch name. For example, it might include a rationale of WHY the change was necessary. By default Darcs asks if you want to add a description; the –edit-long-comment and –skip-long-comment can be used to answer 'yes' or 'no' (respectively) to this prompt. Finally, the –logfile option allows you to supply a file that already contains the patch name (first line) and patch description (subsequent lines). This is useful if a previous record failed and left a darcs-record-0 file.

Each patch is attributed to its author, usually by email address (for example, 'Fred Bloggs ¡fred@example.net¿'). Darcs looks in several places for this author string: the –author option, the files _darcs/prefs/author (in the repository) and /.darcs/author (in your home directory), and the environment variables $DARCS_EMAIL and $EMAIL. If none of those exist, Darcs will prompt you for an author string and write it to _darcs/prefs/author. Note that if if you have more than one email address, note that you can put them all in /.darcs/author, one author per line. Darcs will still prompt you for an author, but it allows you to select from the list, or to type in an alternative.

The patch date is generated automatically. It can only be spoofed by using the –pipe option.

If a test command has been defined with 'darcs setpref', attempting to record a patch will cause the test command to be run in a clean copy of the working tree (that is, including only recorded changes). If the test fails, you will be offered to abort the record operation.

The –set-scripts-executable option causes scripts to be made executable in the clean copy of the working tree, prior to running the test. See 'darcs get' for an explanation of the script heuristic.

If your test command is tediously slow (e.g. 'make all') and you are recording several patches in a row, you may wish to use –no-test to skip all but the final test.

To see some context (unchanged lines) around each change, use the –unified option.

```
Usage: darcs record [OPTION]... [FILE or DIRECTORY]...
```

Options:

```
-m  --name PATCHNAME                          name of patch
-A  --author EMAIL                            specify author id
    --test                                    run the test script
    --no-test                                 don't run the test script
    --leave-test-directory                    don't remove the test directory
    --remove-test-directory                   remove the test directory
-a  --all                                     answer yes to all patches
    --pipe                                    ask user interactively for the patch metadata
-i  --interactive                             prompt user interactively
    --ask-deps                                ask for extra dependencies
    --no-ask-deps                             don't ask for extra dependencies
    --edit-long-comment                       edit the long comment by default
    --skip-long-comment                       don't give a long comment
    --prompt-long-comment                     prompt for whether to edit the long comment
-l  --look-for-adds                           look for (non-boring) files that could be
                                              added
    --dont-look-for-adds,--no-look-for-adds   don't look for any files that could be added
                                              [DEFAULT]
    --repodir DIRECTORY                       specify the repository directory in which to
                                              run
-u  --unified                                 output changes in a darcs-specific format
                                              similar to diff -u
    --no-unified                              output changes in darcs' usual format
```

Advanced options:

```
    --logfile FILE                                    give patch name and comment in file
    --delete-logfile                                  delete the logfile when done
    --no-delete-logfile                               keep the logfile when done [DEFAULT]
    --compress                                        create compressed patches
    --dont-compress,--no-compress                     don't create compressed patches
    --ignore-times                                    don't trust the file modification times
    --no-ignore-times                                 trust modification times to find modified files
                                                      [DEFAULT]
    --umask UMASK                                     specify umask to use when writing
    --set-scripts-executable                          make scripts executable
    --dont-set-scripts-executable,--no-set-scripts-executable  don't make scripts executable
```

Create a patch from unrecorded changes.

`--ask-deps`

Each patch may depend on any number of previous patches. If you choose to make your patch depend on a previous patch, that patch is required to be applied before your patch can be applied to a repository. This can be used, for example, if a piece of code requires a function to be defined, which was defined in an earlier patch.

If you want to manually define any dependencies for your patch, you can use the `--ask-deps` flag, and darcs will ask you for the patch's dependencies.

It is possible to record a patch which has no actual changes but which has specific dependencies. This type of patch can be thought of as a "partial tag". The `darcs tag` command will record a patch with no actual changes but which depends on the entire current inventory of the repository. The `darcs record --ask-deps` with no selected changes will record a patch that depends on only those patches selected via the `--ask-deps` operation, resulting in a patch which describes a set of patches; the presence of this primary patch in a repository implies the presence of (at least) the depended-upon patches.

`--pipe`

If you run record with the `--pipe` option, you will be prompted for the patch date, author, and the long comment. The long comment will extend until the end of file or stdin is reached (ctrl-D on Unixy systems, ctrl-Z on systems running a Microsoft OS).

This interface is intended for scripting darcs, in particular for writing repository conversion scripts. The prompts are intended mostly as a useful guide (since scripts won't need them), to help you understand the format in which to provide the input. Here's an example of what the `--pipe` prompts look like:

```
What is the date? Mon Nov 15 13:38:01 EST 2004
Who is the author? David Roundy
What is the log? One or more comment lines
```

`--interactive`

By default, `record` works interactively. Probably the only thing you need to know about using this is that you can press `?` at the prompt to be shown a list of the rest of the options and what they do. The rest should be clear from there. Here's a "screenshot" to demonstrate:

```
hunk ./hello.pl +2
+#!/usr/bin/perl
+print "Hello World!\n";
Shall I record this patch? (2/2) [ynWsfqadjk], or ? for help: ?
How to use record...
y: record this patch
n: don't record it
w: wait and decide later, defaulting to no

s: don't record the rest of the changes to this file
f: record the rest of the changes to this file

d: record selected patches
a: record all the remaining patches
q: cancel record

j: skip to next patch
k: back up to previous patch
h or ?: show this help

<Space>: accept the current default (which is capitalized)
```

What you can't see in that "screenshot" is that `darcs` will also try to use color in your terminal to make the output even easier to read.

5.6.2 darcs pull

Pull is used to bring changes made in another repository into the current repository (that is, either the one in the current directory, or the one specified with the –repodir option). Pull allows you to bring over all or some of the patches that are in that repository but not in this one. Pull accepts arguments, which are URLs from which to pull, and when called without an argument, pull will use the repository from which you have most recently either pushed or pulled.

See 'darcs help apply' for detailed description of many options.

`Usage: darcs pull [OPTION]... [REPOSITORY]...`

Options:

	`--matches PATTERN`	select patches matching PATTERN
`-p`	`--patches REGEXP`	select patches matching REGEXP
`-t`	`--tags REGEXP`	select tags matching REGEXP
`-a`	`--all`	answer yes to all patches
`-i`	`--interactive`	prompt user interactively
	`--mark-conflicts`	mark conflicts [DEFAULT]
	`--allow-conflicts`	allow conflicts, but don't mark them
	`--dont-allow-conflicts,--no-allow-conflicts`	fail if there are patches that would create conflicts
	`--skip-conflicts`	filter out any patches that would create conflicts
	`--external-merge COMMAND`	use external tool to merge conflicts
	`--test`	run the test script
	`--no-test`	don't run the test script
	`--dry-run`	don't actually take the action
	`--xml-output`	generate XML formatted output
`-s`	`--summary`	summarize changes
	`--no-summary`	don't summarize changes
	`--no-deps`	don't automatically fulfill dependencies
	`--dont-prompt-for-dependencies`	don't ask about patches that are depended on by matched patches (with –match or –patch)
	`--prompt-for-dependencies`	prompt about patches that are depended on by matched patches [DEFAULT]
	`--set-default`	set default repository
	`--no-set-default`	don't set default repository [DEFAULT]
	`--repodir DIRECTORY`	specify the repository directory in which to run
	`--ignore-unrelated-repos`	do not check if repositories are unrelated

Advanced options:

`--intersection`	take intersection of all repositories
`--union`	take union of all repositories [DEFAULT]
`--complement`	take complement of repositories (in order listed)
`--compress`	create compressed patches
`--dont-compress,--no-compress`	don't create compressed patches
`--ignore-times`	don't trust the file modification times
`--no-ignore-times`	trust modification times to find modified files [DEFAULT]
`--remote-repo URL`	specify the remote repository URL to work with
`--set-scripts-executable`	make scripts executable
`--dont-set-scripts-executable,--no-set-scripts-executable`	don't make scripts executable
`--umask UMASK`	specify umask to use when writing
`--restrict-paths`	don't allow darcs to touch external files or repo metadata
`--dont-restrict-paths,--no-restrict-paths`	allow darcs to modify any file or directory (unsafe)
`--reverse`	show changes in reverse order
`--no-reverse`	show changes in the usual order [DEFAULT]
`--pause-for-gui`	pause for an external diff or merge command to finish [DEFAULT]
`--no-pause-for-gui`	return immediately after external diff or merge command finishes
`--no-http-pipelining`	disable HTTP pipelining
`--remote-darcs COMMAND`	name of the darcs executable on the remote server

Copy and apply patches from another repository to this one.

`--intersection, --union [DEFAULT], --complement`

If you provide more than one repository as an argument to pull, darcs' behavior is determined by the presence of the `--complement`, `--intersection`, and `--union` flags.

- The default (`--union`) behavior is to pull any patches that are in any of the specified repositories.
- If you instead specify the `--intersection` flag, darcs will only pull those patches which are present in all source repositories.
- If you specify the `--complement` flag, darcs will only pull elements in the first repository that do not exist in any of the remaining repositories.

`--external-merge`

You can use an external interactive merge tool to resolve conflicts with the flag `--external-merge`. For more details see subsection 5.1.1.

`--matches, --patches, --tags, --no-deps`

The `--patches`, `--matches`, `--tags`, and `--no-deps` options can be used to select which patches to pull, as described in subsection 5.1.

`--no-test, --test`

If you specify the `--test` option, pull will run the test (if a test exists) on a scratch copy of the repository contents prior to actually performing the pull. If the test fails, the pull will be aborted.

`--verbose`

Adding the `--verbose` option causes another section to appear in the output which also displays a summary of patches that you have and the remote repository lacks. Thus, the following syntax can be used to show you all the patch differences between two repositories:

```
darcs pull --dry-run --verbose
```

5.6.3 darcs push

Push is the opposite of pull. Push allows you to copy changes from the current repository into another repository.

`Usage: darcs push [OPTION]... [REPOSITORY]`

Options:

	`--matches PATTERN`	select patches matching PATTERN
`-p`	`--patches REGEXP`	select patches matching REGEXP
`-t`	`--tags REGEXP`	select tags matching REGEXP
	`--no-deps`	don't automatically fulfill dependencies
	`--dont-prompt-for-dependencies`	don't ask about patches that are depended on by matched patches (with –match or –patch)
	`--prompt-for-dependencies`	prompt about patches that are depended on by matched patches [DEFAULT]
`-a`	`--all`	answer yes to all patches
`-i`	`--interactive`	prompt user interactively
	`--sign`	sign the patch with your gpg key
	`--sign-as KEYID`	sign the patch with a given keyid
	`--sign-ssl IDFILE`	sign the patch using openssl with a given private key
	`--dont-sign,--no-sign`	don't sign the patch
	`--dry-run`	don't actually take the action
	`--xml-output`	generate XML formatted output
`-s`	`--summary`	summarize changes
	`--no-summary`	don't summarize changes
	`--repodir DIRECTORY`	specify the repository directory in which to run
	`--set-default`	set default repository
	`--no-set-default`	don't set default repository [DEFAULT]
	`--ignore-unrelated-repos`	do not check if repositories are unrelated

Advanced options:

`--apply-as USERNAME`	apply patch as another user using sudo
`--no-apply-as`	don't use sudo to apply as another user [DEFAULT]
`--remote-repo URL`	specify the remote repository URL to work with
`--reverse`	show changes in reverse order
`--no-reverse`	show changes in the usual order [DEFAULT]
`--no-http-pipelining`	disable HTTP pipelining
`--remote-darcs COMMAND`	name of the darcs executable on the remote server

Copy and apply patches from this repository to another one.

For obvious reasons, you can only push to repositories to which you have write access. In addition, you can only push to repos that you access either on the local file system or with ssh. In order to apply with ssh, darcs must also be installed on the remote computer. The command invoked to run ssh may be configured by the `DARCS_SSH` environment variable (see subsection 3.4). The command invoked via ssh is always `darcs`, i.e. the darcs executable must be in the default path on the remote machine.

Push works by creating a patch bundle, and then running darcs apply in the target repository using that patch bundle. This means that the default options for *apply* in the *target* repository (such as, for example, `--test`) will affect the behavior of push. This also means that push is somewhat less efficient than pull.

When you receive an error message such as

```
bash: darcs: command not found
```

then this means that the darcs on the remote machine could not be started. Make sure that the darcs executable is called `darcs` and is found in the default path. The default path can be different in interactive and in non-interactive shells. Say

```
ssh login@remote.machine darcs
```

to try whether the remote darcs can be found, or

```
ssh login@remote.machine 'echo $PATH'
```

(note the single quotes) to check the default path.

`--apply-as`

If you give the `--apply-as` flag, darcs will use sudo to apply the changes as a different user. This can be useful if you want to set up a system where several users can modify the same repository, but you don't want to allow them full write access. This isn't secure against skilled malicious attackers, but at least can protect your repository from clumsy, inept or lazy users.

`--matches, --patches, --tags, --no-deps`

The `--patches`, `--matches`, `--tags`, and `--no-deps` options can be used to select which patches to push, as described in subsection 5.1.

When there are conflicts, the behavior of push is determined by the default flags to `apply` in the *target* repository. Most commonly, for pushed-to repositories, you'd like to have `--dont-allow-conflicts` as a default option to apply (by default, it is already the default...). If this is the case, when there are conflicts on push, darcs will fail with an error message. You can then resolve by pulling the conflicting patch, recording a resolution and then pushing the resolution together with the conflicting patch.

Darcs does not have an explicit way to tell you which patch conflicted, only the file name. You may want to pull all the patches from the remote repository just to be sure. If you don't want to do this in your working directory, you can create another darcs working directory for this purpose.

If you want, you could set the target repository to use `--allow-conflicts`. In this case conflicting patches will be applied, but the conflicts will not be marked in the working directory.

If, on the other hand, you have `--mark-conflicts` specified as a default flag for apply in the target repository, when there is a conflict, it will be marked in the working directory of the target repository. In this case, you should resolve the conflict in the target repository itself.

5.6.4 darcs send

Send is used to prepare a bundle of patches that can be applied to a target repository. Send accepts the URL of the repository as an argument. When called without an argument, send will use the most recent repository that was either pushed to, pulled from or sent to. By default, the patch bundle is sent by email, although you may save it to a file.

```
Usage: darcs send [OPTION]... [REPOSITORY]
```

Options:

```
     --matches PATTERN                          select patches matching PATTERN
 -p  --patches REGEXP                           select patches matching REGEXP
 -t  --tags REGEXP                              select tags matching REGEXP
     --no-deps                                  don't automatically fulfill dependencies
     --dont-prompt-for-dependencies             don't ask about patches that are depended on
                                                by matched patches (with –match or –patch)
     --prompt-for-dependencies                  prompt about patches that are depended on
                                                by matched patches [DEFAULT]
 -a  --all                                      answer yes to all patches
 -i  --interactive                              prompt user interactively
     --from EMAIL                               specify email address
 -A  --author EMAIL                             specify author id
     --to EMAIL                                 specify destination email
     --cc EMAIL                                 mail results to additional EMAIL(s)
     --subject SUBJECT                          specify mail subject
     --in-reply-to EMAIL                        specify in-reply-to header
     --charset CHARSET                          specify mail charset
 -o  --output FILE                              specify output filename
 -O  --output-auto-name[=DIRECTORY]             output to automatically named file in
                                                DIRECTORY, default: current directory
     --sign                                     sign the patch with your gpg key
     --sign-as KEYID                            sign the patch with a given keyid
     --sign-ssl IDFILE                          sign the patch using openssl with a given
                                                private key
     --dont-sign,--no-sign                      don't sign the patch
     --dry-run                                  don't actually take the action
     --xml-output                               generate XML formatted output
 -s  --summary                                  summarize changes
     --no-summary                               don't summarize changes
     --edit-description                         edit the patch bundle description
                                                [DEFAULT]
     --dont-edit-description,--no-edit-description  don't edit the patch bundle description
     --set-default                              set default repository
     --no-set-default                           don't set default repository [DEFAULT]
     --repodir DIRECTORY                        specify the repository directory in which to
                                                run
     --sendmail-command COMMAND                 specify sendmail command
     --ignore-unrelated-repos                   do not check if repositories are unrelated
```

Advanced options:

`--logfile FILE`	give patch name and comment in file
`--delete-logfile`	delete the logfile when done
`--no-delete-logfile`	keep the logfile when done [DEFAULT]
`--remote-repo URL`	specify the remote repository URL to work with
`--context FILENAME`	send to context stored in FILENAME
`--reverse`	show changes in reverse order
`--no-reverse`	show changes in the usual order [DEFAULT]
`--no-http-pipelining`	disable HTTP pipelining
`--remote-darcs COMMAND`	name of the darcs executable on the remote server

Send by email a bundle of one or more patches.

Do not confuse the `--author` options with the return address that `darcs send` will set for your patch bundle.

For example, if you have two email addresses A and B:

If you use `--author A` but your machine is configured to send mail from address B by default, then the return address on your message will be B.

If you use `--from A` and your mail client supports setting the From: address arbitrarily (some non-Unix-like mail clients, especially, may not support this), then the return address will be A; if it does not support this, then the return address will be B.

If you supply neither `--from` nor `--author`, then the return address will be B.

In addition, unless you specify the sendmail command with `--sendmail-command`, darcs sends email using the default email command on your computer. This default command is determined by the `configure` script. Thus, on some non-Unix-like OSes, `--from` is likely to not work at all.

`--output, --to, --cc`

The `--output`, `--output-auto-name`, and `--to` flags determine what darcs does with the patch bundle after creating it. If you provide an `--output` argument, the patch bundle is saved to that file. If you specify `--output-auto-name`, the patch bundle is saved to a file with an automatically generated name. If you give one or more `--to` arguments, the bundle of patches is sent to those locations. The locations may either be email addresses or urls that the patch should be submitted to via HTTP.

If you don't provide any of these options, darcs will look at the contents of the `_darcs/prefs/email` file in the target repository (if it exists), and send the patch by email to that address. In this case, you may use the `--cc` option to specify additional recipients without overriding the default repository email address.

If `_darcs/prefs/post` exists in the target repository, darcs will upload to the URL contained in that file, which may either be a `mailto:` URL, or an `http://` URL. In the latter case, the patch is posted to that URL.

If there is no email address associated with the repository, darcs will prompt you for an email address.

`--subject`

Use the `--subject` flag to set the subject of the e-mail to be sent. If you don't provide a subject on the command line, darcs will make one up based on names of the patches in the patch bundle.

`--in-reply-to`

Use the `--in-reply-to` flag to set the In-Reply-To and References headers of the e-mail to be sent. By default no additional headers are included so e-mail will not be treated as reply by mail readers.

`--matches, --patches, --tags, --no-deps`

The `--patches`, `--matches`, `--tags`, and `--no-deps` options can be used to select which patches to send, as described in subsection 5.1.

`--edit-description`

If you want to include a description or explanation along with the bundle of patches, you need to specify the `--edit-description` flag, which will cause darcs to open up an editor with which you can compose a message to go along with your patches.

`--sendmail-command`

If you want to use a command different from the default one for sending email, you need to specify a command line with the `--sendmail-command` option. The command line can contain some format specifiers which are replaced by the actual values. Accepted format specifiers are `%s` for subject, `%t` for to, `%c` for cc, `%b` for the body of the mail, `%f` for from, `%a` for the patch bundle and the same specifiers in uppercase for the URL-encoded values. Additionally you can add `%<` to the end of the command line if the command expects the complete email message on standard input. E.g. the command lines for evolution and msmtp look like this:

```
evolution "mailto:%T?subject=%S&attach=%A&cc=%C&body=%B"
msmtp -t %<
```

5.6.5 darcs apply

The 'darcs apply' command takes a patch bundle and attempts to insert it into the current repository. In addition to invoking it directly on bundles created by 'darcs send', it is used internally by 'darcs push' and 'darcs put' on the remote end of an SSH connection.

If no file is supplied, the bundle is read from standard input.

If given an email instead of a patch bundle, Darcs will look for the bundle as a MIME attachment to that email. Currently this will fail if the MIME boundary is rewritten, such as in Courier and Mail.app.

If the '–reply noreply@example.net' option is used, and the bundle is attached to an email, Darcs will send a report (indicating success or failure) to the sender of the bundle (the To field). The argument to noreply is the address the report will appear to originate FROM.

The –cc option will cause the report to be CC'd to another address, for example '–cc reports@lists.example.net,admin@lists. Using –cc without –reply is undefined.

If gpg(1) is installed, you can use '–verify pubring.gpg' to reject bundles that aren't signed by a key in pubring.gpg.

If –test is supplied and a test is defined (see 'darcs setpref'), the bundle will be rejected if the test fails after applying it. In that case, the rejection email from –reply will include the test output.

A patch bundle may introduce unresolved conflicts with existing patches or with the working tree. By default, Darcs will add conflict markers (see 'darcs mark-conflicts').

The –external-merge option lets you resolve these conflicts using an external merge tool. In the option, '%a' is replaced with the common ancestor (merge base), '%1' with the first version, '%2' with the second version, and '%o' with the path where your resolved content should go. For example, to use the xxdiff visual merge tool you'd specify: –external-merge='xxdiff -m -O -M %o %1 %a %2'

The –allow-conflicts option will skip conflict marking; this is useful when you want to treat a repository as just a bunch of patches, such as using 'darcs pull –union' to download of your co-workers patches before going offline.

This can mess up unrecorded changes in the working tree, forcing you to resolve the conflict immediately. To simply reject bundles that introduce unresolved conflicts, using the –dont-allow-conflicts option. Making this the default in push-based workflows is strongly recommended.

Unlike most Darcs commands, 'darcs apply' defaults to –all. Use the –interactive option to pick which patches to apply from a bundle.

`Usage: darcs apply [OPTION]... <PATCHFILE>`

Options:

	`--verify PUBRING`	verify that the patch was signed by a key in PUBRING
	`--verify-ssl KEYS`	verify using openSSL with authorized keys from file KEYS
	`--no-verify`	don't verify patch signature
`-a`	`--all`	answer yes to all patches
`-i`	`--interactive`	prompt user interactively
	`--dry-run`	don't actually take the action
	`--xml-output`	generate XML formatted output
	`--matches PATTERN`	select patches matching PATTERN
`-p`	`--patches REGEXP`	select patches matching REGEXP
`-t`	`--tags REGEXP`	select tags matching REGEXP
	`--mark-conflicts`	mark conflicts
	`--allow-conflicts`	allow conflicts, but don't mark them
	`--no-resolve-conflicts`	equivalent to –dont-allow-conflicts, for backwards compatibility
	`--dont-allow-conflicts,--no-allow-conflicts`	fail if there are patches that would create conflicts [DEFAULT]
	`--skip-conflicts`	filter out any patches that would create conflicts
	`--external-merge COMMAND`	use external tool to merge conflicts
	`--no-test`	don't run the test script
	`--test`	run the test script
	`--leave-test-directory`	don't remove the test directory
	`--remove-test-directory`	remove the test directory
	`--repodir DIRECTORY`	specify the repository directory in which to run

Advanced options:

`--reply FROM`	reply to email-based patch using FROM address
`--cc EMAIL`	mail results to additional EMAIL(s). Requires –reply
`--happy-forwarding`	forward unsigned messages without extra header
`--no-happy-forwarding`	don't forward unsigned messages without extra header [DEFAULT]
`--sendmail-command COMMAND`	specify sendmail command
`--ignore-times`	don't trust the file modification times
`--no-ignore-times`	trust modification times to find modified files [DEFAULT]
`--compress`	create compressed patches
`--dont-compress,--no-compress`	don't create compressed patches
`--set-scripts-executable`	make scripts executable
`--dont-set-scripts-executable,--no-set-scripts-executable`	don't make scripts executable
`--umask UMASK`	specify umask to use when writing
`--restrict-paths`	don't allow darcs to touch external files or repo metadata
`--dont-restrict-paths,--no-restrict-paths`	allow darcs to modify any file or directory (unsafe)
`--reverse`	show changes in reverse order
`--no-reverse`	show changes in the usual order [DEFAULT]
`--pause-for-gui`	pause for an external diff or merge command to finish [DEFAULT]
`--no-pause-for-gui`	return immediately after external diff or merge command finishes

Apply a patch bundle created by 'darcs send'.

Darcs apply accepts a single argument, which is the name of the patch file to be applied. If you omit this argument, the patch is read from standard input. Darcs also interprets an argument of '' to mean it should read the file from standard input. This allows you to use apply with a pipe from your email program, for example.

`--verify`

`--external-merge`

You can use an external interactive merge tool to resolve conflicts with the flag `--external-merge`. For more details see subsection 5.1.1.

`--sendmail-command`

If you want to use a command different from the default one for sending mail, you need to specify a command line with the `--sendmail-command` option. The command line can contain the format specifier `%t` for to and you can add `%<` to the end of the command line if the command expects the complete mail on standard input. For example, the command line for msmtp looks like this:

```
msmtp -t %<
```

5.7 Seeing what you've done

5.7.1 darcs whatsnew

The 'darcs whatsnew' command lists unrecorded changes to the working tree. If you specify a set of files and directories, only unrecorded changes to those files and directories are listed.

With the –summary option, the changes are condensed to one line per file, with mnemonics to indicate the nature and extent of the change. The –look-for-adds option causes candidates for 'darcs add' to be included in the summary output. Summary mnemonics are as follows:

'A f' and 'A d/' respectively mean an added file or directory. 'R f' and 'R d/' respectively mean a removed file or directory. 'M f -N +M rP' means a modified file, with N lines deleted, M lines added, and P lexical replacements. 'f -¿ g' means a moved file or directory. 'a f' and 'a d/' respectively mean a new, but unadded, file or directory, when using –look-for-adds.

An exclamation mark (!) as in 'R! foo.c', means the hunk is known to conflict with a hunk in another patch. The phrase 'duplicated' means the hunk is known to be identical to a hunk in another patch.

By default, 'darcs whatsnew' uses Darcs' internal format for changes. To see some context (unchanged lines) around each change, use the –unified option. To view changes in conventional 'diff' format, use the 'darcs diff' command; but note that 'darcs whatsnew' is faster.

This command exits unsuccessfully (returns a non-zero exit status) if there are no unrecorded changes.

```
Usage: darcs whatsnew [OPTION]... [FILE or DIRECTORY]...
```

Options:

`-s`	`--summary`	summarize changes
	`--no-summary`	don't summarize changes
`-u`	`--unified`	output changes in a darcs-specific format similar to diff -u
	`--no-unified`	output changes in darcs' usual format
`-l`	`--look-for-adds`	look for (non-boring) files that could be added
	`--dont-look-for-adds,--no-look-for-adds`	don't look for any files that could be added [DEFAULT]
	`--repodir DIRECTORY`	specify the repository directory in which to run

Advanced options:

`--ignore-times`	don't trust the file modification times
`--no-ignore-times`	trust modification times to find modified files [DEFAULT]
`--boring`	don't skip boring files
`--no-boring`	skip boring files [DEFAULT]

List unrecorded changes in the working tree.

5.7.2 darcs changes

The 'darcs changes' command lists the patches that constitute the current repository or, with –repo, a remote repository. Without options or arguments, ALL patches will be listed.

When given one or more files or directories as arguments, only patches which affect those files or directories are listed. This includes changes that happened to files before they were moved or renamed.

When given a –from-tag, –from-patch or –from-match, only changes since that tag or patch are listed. Similarly, the –to-tag, –to-patch and –to-match options restrict the list to older patches.

The –last and –max-count options both limit the number of patches listed. The former applies BEFORE other filters, whereas the latter applies AFTER other filters. For example 'darcs changes foo.c –max-count 3' will print the last three patches that affect foo.c, whereas 'darcs changes –last 3 foo.c' will, of the last three patches, print only those that affect foo.c.

Three output formats exist. The default is –human-readable. You can also select –context, which is the internal format (as seen in patch bundles) that can be re-read by Darcs (e.g. 'darcs get –context').

Finally, there is –xml-output, which emits valid XML... unless a the patch metadata (author, name or description) contains a non-ASCII character and was recorded in a non-UTF8 locale.

Note that while the –context flag may be used in conjunction with –xml-output or –human-readable, in neither case will darcs get be able to read the output. On the other hand, sufficient information WILL be output for a knowledgeable human to recreate the current state of the repository.

```
Usage: darcs changes [OPTION]... [FILE or DIRECTORY]...
```

Options:

	`--to-match PATTERN`	select changes up to a patch matching PATTERN
	`--to-patch REGEXP`	select changes up to a patch matching REGEXP
	`--to-tag REGEXP`	select changes up to a tag matching REGEXP
	`--from-match PATTERN`	select changes starting with a patch matching PATTERN
	`--from-patch REGEXP`	select changes starting with a patch matching REGEXP
	`--from-tag REGEXP`	select changes starting with a tag matching REGEXP
	`--last NUMBER`	select the last NUMBER patches
`-n`	`--index N-M`	select a range of patches
	`--matches PATTERN`	select patches matching PATTERN
`-p`	`--patches REGEXP`	select patches matching REGEXP
`-t`	`--tags REGEXP`	select tags matching REGEXP
	`--max-count NUMBER`	return only NUMBER results
	`--only-to-files`	show only changes to specified files
	`--no-only-to-files`	show changes to all files [DEFAULT]
	`--context`	give output suitable for get –context
	`--xml-output`	generate XML formatted output
	`--human-readable`	give human-readable output
	`--number`	number the changes
	`--count`	output count of changes
`-s`	`--summary`	summarize changes
	`--no-summary`	don't summarize changes
	`--reverse`	show changes in reverse order
	`--no-reverse`	show changes in the usual order [DEFAULT]
	`--repo URL`	specify the repository URL
	`--repodir DIRECTORY`	specify the repository directory in which to run
`-a`	`--all`	answer yes to all patches
`-i`	`--interactive`	prompt user interactively

Advanced options:

`--no-http-pipelining`	disable HTTP pipelining
`--remote-darcs COMMAND`	name of the darcs executable on the remote server

List patches in the repository.

5.8 More advanced commands

5.8.1 darcs tag

The 'darcs tag' command names the current repository state, so that it can easily be referred to later. Every 'important' state should be tagged; in particular it is good practice to tag each stable release with a number or codename. Advice on release numbering can be found at http://producingoss.com/en/development-cycle.html.

To reproduce the state of a repository 'R' as at tag 't', use the command 'darcs get –tag t R'. The command 'darcs show tags' lists all tags in the current repository.

Tagging also provides significant performance benefits: when Darcs reaches a shared tag that depends on all antecedent patches, it can simply stop processing.

Like normal patches, a tag has a name, an author, a timestamp and an optional long description, but it does not change the working tree. A tag can have any name, but it is generally best to pick a naming scheme and stick to it.

The 'darcs tag' command accepts the –pipe option, which behaves as described in 'darcs record'.

```
Usage: darcs tag [OPTION]... [TAGNAME]
```

Options:

`-m`	`--name PATCHNAME`	name of patch
`-A`	`--author EMAIL`	specify author id
	`--pipe`	ask user interactively for the patch metadata
`-i`	`--interactive`	prompt user interactively
	`--edit-long-comment`	edit the long comment by default
	`--skip-long-comment`	don't give a long comment
	`--prompt-long-comment`	prompt for whether to edit the long comment
	`--repodir DIRECTORY`	specify the repository directory in which to run

Advanced options:

`--compress`	create compressed patches
`--dont-compress,--no-compress`	don't create compressed patches
`--umask UMASK`	specify umask to use when writing

Name the current repository state for future reference.

5.8.2 darcs setpref

When working on project with multiple repositories and contributors, it is sometimes desirable for a preference to be set consistently project-wide. This is achieved by treating a preference set with 'darcs setpref' as an unrecorded change, which can then be recorded and then treated like any other patch.

Valid preferences are:

test – a shell command that runs regression tests predist – a shell command to run before 'darcs dist' boringfile – the path to a version-controlled boring file binariesfile – the path to a version-controlled binaries file

For example, a project using GNU autotools, with a 'make test' target to perform regression tests, might enable Darcs' integrated regression testing with the following command:

darcs setpref test 'autoconf && ./configure && make && make test'

Note that merging is not currently implemented for preferences: if two patches attempt to set the same preference, the last patch applied to the repository will always take precedence. This is considered a low-priority bug, because preferences are seldom set.

```
Usage: darcs setpref [OPTION]... <PREF> <VALUE>
```

Options:

`--repodir DIRECTORY`	specify the repository directory in which to run

Advanced options:

`--umask UMASK`	specify umask to use when writing

Set a preference (test, predist, boringfile or binariesfile).

5.8.3 darcs test

If a regression test is defined (see 'darcs setpref') it will be run.

```
Usage: darcs test [OPTION]...
```

Options:

```
--leave-test-directory      don't remove the test directory
--remove-test-directory     remove the test directory
--repodir DIRECTORY         specify the repository directory in which to
                            run
```

Run regression test.

If you like, you can configure your repository to be able to run a test suite of some sort. You can do this by using "setpref" to set the "test" value to be a command to run, e.g.

```
% darcs setpref test "sh configure && make && make test"
```

Or, if you want to define a test specific to one copy of the repository, you could do this by editing the file `_darcs/prefs/prefs`.

`--leave-test-directory, --remove-test-directory`

Normally darcs deletes the directory in which the test was run afterwards. Sometimes (especially when the test fails) you'd prefer to be able to be able to examine the test directory after the test is run. You can do this by specifying the `--leave-test-directory` flag. Alas, there is no way to make darcs leave the test directory only if the test fails. The opposite of `--leave-test-directory` is `--remove-test-directory`, which could come in handy if you choose to make `--leave-test-directory` the default (see section 3.1).

5.8.4 darcs repair

The 'darcs repair' command attempts to fix corruption in the current repository. Currently it can only repair damage to the pristine tree, which is where most corruption occurs.

```
Usage: darcs repair [OPTION]...
```

Options:

```
--repodir DIRECTORY         specify the repository directory in which to
                            run
```

Advanced options:

```
--umask UMASK               specify umask to use when writing
```

Repair a corrupted repository.

5.8.5 darcs optimize

The 'darcs optimize' command modifies the current repository in an attempt to reduce its resource requirements. By default a single fast, safe optimization is performed; additional optimization techniques can be enabled by passing options to 'darcs optimize'.

The default optimization moves recent patches (those not included in the latest tag) to the 'front', reducing the amount that a typical remote command needs to download. It should also reduce the CPU time needed for some operations.

The 'darcs optimize –relink' command hard-links patches that the current repository has in common with its peers. Peers are those repositories listed in _darcs/prefs/sources, or defined with the '–sibling' option (which can be used multiple times).

Darcs uses hard-links automatically, so this command is rarely needed. It is most useful if you used 'cp -r' instead of 'darcs get' to copy a repository, or if you pulled the same patch from a remote repository into multiple local repositories.

By default patches are compressed with zlib (RFC 1951) to reduce storage (and download) size. In exceptional circumstances, it may be preferable to avoid compression. In this case the '–dont-compress' option can be used (e.g. with 'darcs record') to avoid compression.

The 'darcs optimize –uncompress' and 'darcs optimize –compress' commands can be used to ensure existing patches in the current repository are respectively uncompressed or compressed. Note that repositories in the legacy 'old-fashioned-inventory' format have a .gz extension on patch files even when uncompressed.

There is one more optimization which CAN NOT be performed by this command. Every time your record a patch, a new inventory file is written to _darcs/inventories/, and old inventories are never reaped.

If _darcs/inventories/ is consuming a relatively large amount of space, you can safely reclaim it by using 'darcs get' to make a complete copy of the repo. When doing so, don't forget to copy over any unsaved changes you have made to the working tree or to unversioned files in _darcs/prefs/ (such as _darcs/prefs/author).

```
Usage: darcs optimize [OPTION]...
```

Options:

`--repodir DIRECTORY`	specify the repository directory in which to run
`--reorder-patches`	reorder the patches in the repository
`--sibling URL`	specify a sibling directory
`--relink`	relink random internal data to a sibling
`--upgrade`	upgrade repository to latest compatible format
`--pristine`	optimize hashed pristine layout
`--http`	optimize repository for getting over network

Advanced options:

`--compress`	create compressed patches
`--dont-compress,--no-compress`	don't create compressed patches
`--uncompress`	uncompress patches
`--umask UMASK`	specify umask to use when writing

Optimize the repository.

`--reorder-patches`

The `--reorder-patches` option causes Darcs to create an optimal ordering of its internal patch inventory. This may help to produce shorter 'context' lists when sending patches, and may improve performance for some other operations as well. You should not run `--reorder-patches` on a repository from which someone may be simultaneously pulling or getting, as this could lead to repository corruption.

The `--upgrade` option for `darcs optimize` performs an inplace upgrade of your repository to the latest *compatible* format. Right now means that darcs 1 old-fashioned repositories will be upgraded to darcs-1 hashed repositories (and notably, not to darcs 2 repositories as that would not be compatible; see `darcs convert`).

5.9 Undoing, redoing and running in circles

5.9.1 darcs amend-record

Amend-record updates a 'draft' patch with additions or improvements, resulting in a single 'finished' patch. This is better than recording the additions and improvements as separate patches, because then whenever the 'draft' patch is copied between repositories, you would need to make sure all the extra patches are copied, too.

Do not copy draft patches between repositories, because a finished patch cannot be copied into a repository that contains a draft of the same patch. If this has already happened, 'darcs obliterate' can be used to remove the draft patch.

Do not run amend-record in repository that other developers can pull from, because if they pull while an amend-record is in progress, their repository may be corrupted.

When recording a draft patch, it is a good idea to start the name with 'DRAFT:' so that other developers know it is not finished. When finished, remove it with 'darcs amend-record –edit-long-comment'. To change the patch name without starting an editor, use –patch-name.

Like 'darcs record', if you call amend-record with files as arguments, you will only be asked about changes to those files. So to amend a patch to foo.c with improvements in bar.c, you would run:

darcs amend-record –match 'touch foo.c' bar.c

It is usually a bad idea to amend another developer's patch. To make amend-record only ask about your own patches by default, you can add something like 'amend-record match David Roundy' to /.darcs/defaults, where 'David Roundy' is your name.

```
Usage: darcs amend-record [OPTION]... [FILE or DIRECTORY]...
```

Options:

	`--match PATTERN`	select a single patch matching PATTERN
`-p`	`--patch REGEXP`	select a single patch matching REGEXP
	`--test`	run the test script
	`--no-test`	don't run the test script
	`--leave-test-directory`	don't remove the test directory
	`--remove-test-directory`	remove the test directory
`-a`	`--all`	answer yes to all patches
`-i`	`--interactive`	prompt user interactively
`-A`	`--author EMAIL`	specify author id
`-m`	`--name PATCHNAME`	name of patch
	`--ask-deps`	ask for extra dependencies
	`--no-ask-deps`	don't ask for extra dependencies
	`--edit-long-comment`	edit the long comment by default
	`--skip-long-comment`	don't give a long comment
	`--prompt-long-comment`	prompt for whether to edit the long comment
	`--keep-date`	keep the date of the original patch
	`--no-keep-date`	use the current date for the amended patch
`-l`	`--look-for-adds`	look for (non-boring) files that could be added
	`--dont-look-for-adds,--no-look-for-adds`	don't look for any files that could be added [DEFAULT]
	`--repodir DIRECTORY`	specify the repository directory in which to run
	`--add`	add the changes to patch (default)
	`--unrecord`	subtract the changes from patch
`-u`	`--unified`	output changes in a darcs-specific format similar to diff -u
	`--no-unified`	output changes in darcs' usual format

Advanced options:

`--compress`	create compressed patches
`--dont-compress,--no-compress`	don't create compressed patches
`--ignore-times`	don't trust the file modification times
`--no-ignore-times`	trust modification times to find modified files [DEFAULT]
`--umask UMASK`	specify umask to use when writing
`--set-scripts-executable`	make scripts executable
`--dont-set-scripts-executable,--no-set-scripts-executable`	don't make scripts executable

Improve a patch before it leaves your repository.

5.9.2 darcs rollback

Rollback is used to undo the effects of one or more patches without actually deleting them. Instead, it creates a new patch reversing selected portions. of those changes. Unlike obliterate and unrecord (which accomplish a similar goal) rollback is perfectly safe, since it leaves in the repository a record of its changes.

```
Usage: darcs rollback [OPTION]... [FILE or DIRECTORY]...
```

Options:

	`--from-match PATTERN`	select changes starting with a patch matching PATTERN
	`--from-patch REGEXP`	select changes starting with a patch matching REGEXP
	`--from-tag REGEXP`	select changes starting with a tag matching REGEXP
	`--last NUMBER`	select the last NUMBER patches
	`--matches PATTERN`	select patches matching PATTERN
`-p`	`--patches REGEXP`	select patches matching REGEXP
`-t`	`--tags REGEXP`	select tags matching REGEXP
`-a`	`--all`	answer yes to all patches
`-i`	`--interactive`	prompt user interactively
`-A`	`--author EMAIL`	specify author id
`-m`	`--name PATCHNAME`	name of patch
	`--edit-long-comment`	edit the long comment by default
	`--skip-long-comment`	don't give a long comment
	`--prompt-long-comment`	prompt for whether to edit the long comment
	`--no-test`	don't run the test script
	`--test`	run the test script
	`--leave-test-directory`	don't remove the test directory
	`--remove-test-directory`	remove the test directory
	`--repodir DIRECTORY`	specify the repository directory in which to run
	`--record`	record the rollback patch (default)
	`--no-record`	don't record the rollback patch (only roll back in working dir)

Advanced options:

`--compress`	create compressed patches
`--dont-compress,--no-compress`	don't create compressed patches
`--umask UMASK`	specify umask to use when writing

Record a new patch reversing some recorded changes.

5.9.3 darcs unrecord

Unrecord does the opposite of record in that it makes the changes from patches active changes again which you may record or revert later. The working copy itself will not change. Beware that you should not use this command if you are going to re-record the changes in any way and there is a possibility that another user may have already pulled the patch.

`Usage: darcs unrecord [OPTION]...`

Options:

	`--from-match PATTERN`	select changes starting with a patch matching PATTERN
	`--from-patch REGEXP`	select changes starting with a patch matching REGEXP
	`--from-tag REGEXP`	select changes starting with a tag matching REGEXP
	`--last NUMBER`	select the last NUMBER patches
	`--matches PATTERN`	select patches matching PATTERN
`-p`	`--patches REGEXP`	select patches matching REGEXP
`-t`	`--tags REGEXP`	select tags matching REGEXP
	`--no-deps`	don't automatically fulfill dependencies
	`--dont-prompt-for-dependencies`	don't ask about patches that are depended on by matched patches (with –match or –patch)
	`--prompt-for-dependencies`	prompt about patches that are depended on by matched patches [DEFAULT]
`-a`	`--all`	answer yes to all patches
`-i`	`--interactive`	prompt user interactively
	`--repodir DIRECTORY`	specify the repository directory in which to run

Advanced options:

`--compress`	create compressed patches
`--dont-compress,--no-compress`	don't create compressed patches
`--umask UMASK`	specify umask to use when writing
`--reverse`	show changes in reverse order
`--no-reverse`	show changes in the usual order [DEFAULT]

Remove recorded patches without changing the working copy.

Unrecord can be thought of as undo-record. If a record is followed by an unrecord, everything looks like before the record; all the previously unrecorded changes are back, and can be recorded again in a new patch. The unrecorded patch however is actually removed from your repository, so there is no way to record it again to get it back.[5].

If you want to remove the changes from the working copy too (where they otherwise will show up as unrecorded changes again), you'll also need to `darcs revert`. To do unrecord and revert in one go, you can use `darcs obliterate`.

If you don't revert after unrecording, then the changes made by the unrecorded patches are left in your working tree. If these patches are actually from another repository, interaction (either pushes or pulls) with that repository may be massively slowed down, as darcs tries to cope with the fact that you appear to have made a large number of changes that conflict with those present in the other repository. So if you really want to undo the result of a *pull* operation, use obliterate! Unrecord is primarily intended for when you record a patch, realize it needs just one more change, but would rather not have a separate patch for just that one change.

WARNING: Unrecord should not be run when there is a possibility that another user may be pulling from the same repository. Attempting to do so may cause repository corruption.

`--from-match, --from-patch, --from-tag, --last`

Usually you only want to unrecord the latest changes, and almost never would you want to unrecord changes before a tag—you would have to have unrecorded the tag as well to do that. Therefore, and for efficiency, darcs only prompts you for the latest patches, after some optimal tag.

If you do want to unrecord more patches in one go, there are the `--from` and `--last` options to set the earliest patch selectable to unrecord.

[5]The patch file itself is not actually deleted, but its context is lost, so it cannot be reliably read—your only choice would be to go in by hand and read its contents.

```
--matches, --patches, --tags, --no-deps
```

The `--patches`, `--matches`, `--tags`, and `--no-deps` options can be used to select which patches to unrecord, as described in subsection 5.1.

With these options you can specify what patch or patches to be prompted for by unrecord. This is especially useful when you want to unrecord patches with dependencies, since all the dependent patches (but no others) will be included in the choices. Or if you use `--no-deps` you won't be asked about patches that can't be unrecorded due to depending patches.

Selecting patches can be slow, so darcs cuts the search at the last optimized tag. Use the `--from` or `--last` options to search more or fewer patches.

5.9.4 darcs obliterate

Obliterate completely removes recorded patches from your local repository. The changes will be undone in your working copy and the patches will not be shown in your changes list anymore. Beware that you can lose precious code by obliterating!

```
Usage: darcs obliterate [OPTION]...
```

Options:

	`--from-match PATTERN`	select changes starting with a patch matching PATTERN
	`--from-patch REGEXP`	select changes starting with a patch matching REGEXP
	`--from-tag REGEXP`	select changes starting with a tag matching REGEXP
	`--last NUMBER`	select the last NUMBER patches
	`--matches PATTERN`	select patches matching PATTERN
`-p`	`--patches REGEXP`	select patches matching REGEXP
`-t`	`--tags REGEXP`	select tags matching REGEXP
	`--no-deps`	don't automatically fulfill dependencies
	`--dont-prompt-for-dependencies`	don't ask about patches that are depended on by matched patches (with –match or –patch)
	`--prompt-for-dependencies`	prompt about patches that are depended on by matched patches [DEFAULT]
`-a`	`--all`	answer yes to all patches
`-i`	`--interactive`	prompt user interactively
	`--repodir DIRECTORY`	specify the repository directory in which to run
`-s`	`--summary`	summarize changes
	`--no-summary`	don't summarize changes
`-o`	`--output FILE`	specify output filename
`-O`	`--output-auto-name[=DIRECTORY]`	output to automatically named file in DIRECTORY, default: current directory
	`--dry-run`	don't actually take the action
	`--xml-output`	generate XML formatted output

Advanced options:

`--compress`	create compressed patches
`--dont-compress,--no-compress`	don't create compressed patches
`--ignore-times`	don't trust the file modification times
`--no-ignore-times`	trust modification times to find modified files [DEFAULT]
`--umask UMASK`	specify umask to use when writing
`--reverse`	show changes in reverse order
`--no-reverse`	show changes in the usual order [DEFAULT]

Delete selected patches from the repository. (UNSAFE!)

Obliterate deletes a patch from the repository *and* removes those changes from the working directory. It is therefore a *very dangerous* command. When there are no local changes, obliterate is equivalent to an unrecord followed by a revert, except that revert can be unreverted. In the case of tags, obliterate removes the tag itself, not any other patches.

Note that unpull was the old name for obliterate. Unpull is still an hidden alias for obliterate.

WARNING: Obliterate should not be run when there is a possibility that another user may be pulling from the same repository. Attempting to do so may cause repository corruption.

`--from-match, --from-patch, --from-tag, --last`

Usually you only want to obliterate the latest changes, and almost never would you want to obliterate changes before a tag—you would have to have obliterated the tag as well to do that. Therefore, and for efficiency, darcs only prompts you for the latest patches, after some optimal tag.

If you do want to obliterate more patches in one go, there are the `--from` and `--last` options to set the earliest patch selectable to obliterate.

`--matches, --patches, --tags, --no-deps`

The `--patches`, `--matches`, `--tags`, and `--no-deps` options can be used to select which patches to obliterate, as described in subsection 5.1.

With these options you can specify what patch or patches to be prompted for by obliterate. This is especially useful when you want to obliterate patches with dependencies, since all the dependent patches (but no others) will be included in the choices. Or if you use `--no-deps` you won't be asked about patches that can't be obliterated due to depending patches.

Selecting patches can be slow, so darcs cuts the search at the last optimized tag. Use the `--from` or `--last` options to search more or fewer patches.

5.9.5 darcs revert

The 'darcs revert' command discards unrecorded changes the working tree. As with 'darcs record', you will be asked which hunks (changes) to revert. The –all switch can be used to avoid such prompting. If files or directories are specified, other parts of the working tree are not reverted.

In you accidentally reverted something you wanted to keep (for example, typing 'darcs rev -a' instead of 'darcs rec -a'), you can immediately run 'darcs unrevert' to restore it. This is only guaranteed to work if the repository has not changed since 'darcs revert' ran.

`Usage: darcs revert [OPTION]... [FILE or DIRECTORY]...`

Options:

`-a`	`--all`	answer yes to all patches
`-i`	`--interactive`	prompt user interactively
`-u`	`--unified`	output changes in a darcs-specific format similar to diff -u
	`--no-unified`	output changes in darcs' usual format
	`--repodir DIRECTORY`	specify the repository directory in which to run

Advanced options:

`--ignore-times`	don't trust the file modification times
`--no-ignore-times`	trust modification times to find modified files [DEFAULT]
`--umask UMASK`	specify umask to use when writing

Discard unrecorded changes.

5.9.6 darcs unrevert

Unrevert is a rescue command in case you accidentally reverted something you wanted to keep (for example, typing 'darcs rev -a' instead of 'darcs rec -a').

This command may fail if the repository has changed since the revert took place. Darcs will ask for confirmation before executing an interactive command that will DEFINITELY prevent unreversion.

`Usage: darcs unrevert [OPTION]...`

Options:

	`--ignore-times`	don't trust the file modification times
	`--no-ignore-times`	trust modification times to find modified files [DEFAULT]
`-a`	`--all`	answer yes to all patches
`-i`	`--interactive`	prompt user interactively
	`--repodir DIRECTORY`	specify the repository directory in which to run
`-u`	`--unified`	output changes in a darcs-specific format similar to diff -u
	`--no-unified`	output changes in darcs' usual format

Advanced options:

`--umask UMASK`	specify umask to use when writing

Undo the last revert (may fail if changes after the revert).

5.10 Advanced examination of the repository

5.10.1 darcs diff

The 'darcs diff' command compares two versions of the working tree of the current repository. Without options, the pristine (recorded) and unrecorded working trees are compared. This is lower-level than the 'darcs whatsnew' command, since it outputs a line-by-line diff, and it is also slower. As with 'darcs whatsnew', if you specify files or directories, changes to other files are not listed. The command always uses an external diff utility.

With the –patch option, the comparison will be made between working trees with and without that patch. Patches 'after' the selected patch are not present in either of the compared working trees. The –from-patch and –to-patch options allow the set of patches in the 'old' and 'new' working trees to be specified separately.

The associated tag and match options are also understood, e.g. 'darcs diff –from-tag 1.0 –to-tag 1.1'. All these options assume an ordering of the patch set, so results may be affected by operations such as 'darcs optimize –reorder'.

diff(1) is called with the arguments -rN. The –unified option causes -u to be passed to diff(1). An additional argument can be passed using –diff-opts, such as –diff-opts=-ud or –diff-opts=-wU9.

The –diff-command option can be used to specify an alternative utility, such as meld (GNOME) or opendiff (OS X). Arguments may be included, separated by whitespace. The value is not interpreted by a shell, so shell constructs cannot be used. The arguments %1 and %2 MUST be included, these are substituted for the two working trees being compared. If this option is used, –diff-opts is ignored.

`Usage: darcs diff [OPTION]... [FILE or DIRECTORY]...`

Options:

	`--to-match PATTERN`	select changes up to a patch matching PATTERN
	`--to-patch REGEXP`	select changes up to a patch matching REGEXP
	`--to-tag REGEXP`	select changes up to a tag matching REGEXP
	`--from-match PATTERN`	select changes starting with a patch matching PATTERN
	`--from-patch REGEXP`	select changes starting with a patch matching REGEXP
	`--from-tag REGEXP`	select changes starting with a tag matching REGEXP
	`--match PATTERN`	select a single patch matching PATTERN
`-p`	`--patch REGEXP`	select a single patch matching REGEXP
	`--last NUMBER`	select the last NUMBER patches
`-n`	`--index N-M`	select a range of patches
	`--diff-command COMMAND`	specify diff command (ignores –diff-opts)
	`--diff-opts OPTIONS`	options to pass to diff
`-u`	`--unified`	pass -u option to diff [DEFAULT]
	`--no-unified`	output patch in diff's dumb format
	`--repodir DIRECTORY`	specify the repository directory in which to run
	`--store-in-memory`	do patch application in memory rather than on disk
	`--no-store-in-memory`	do patch application on disk [DEFAULT]

Advanced options:

`--pause-for-gui`	pause for an external diff or merge command to finish [DEFAULT]
`--no-pause-for-gui`	return immediately after external diff or merge command finishes

Create a diff between two versions of the repository.

5.10.2 darcs annotate

The 'darcs annotate' command provides two unrelated operations. When called on a file, it will find the patch that last modified each line in that file. When called on a patch (e.g. using –patch), it will print the internal representation of that patch.

The –summary option will result in a summarized patch annotation, similar to 'darcs whatsnew'. It has no effect on file annotations.

By default, output is in a human-readable format. The –machine-readable option can be used to generate output for machine postprocessing.

`Usage: darcs annotate [OPTION]... [FILE or DIRECTORY]...`

Options:

`-s`	`--summary`	summarize changes
	`--no-summary`	don't summarize changes
`-u`	`--unified`	output changes in a darcs-specific format similar to diff -u
	`--no-unified`	output changes in darcs' usual format
	`--machine-readable`	give machine-readable output
	`--match PATTERN`	select a single patch matching PATTERN
`-p`	`--patch REGEXP`	select a single patch matching REGEXP
`-t`	`--tag REGEXP`	select tag matching REGEXP
`-n`	`--index N`	select one patch
	`--creator-hash HASH`	specify hash of creator patch (see docs)
	`--repodir DIRECTORY`	specify the repository directory in which to run

Display which patch last modified something.

Giving the `--unified` flag implies `--human-readable`, and causes the output to remain in a darcs-specific format that is similar to that produced by `diff --unified`.

If a directory name is given, annotate will output details of the last modifying patch for each file in the directory and the directory itself. The details look like this:

```
# Created by [bounce handling patch
# mark**20040526202216]  as ./test/m7/bounce_handling.pl
   bounce_handling.pl
```

If a patch name and a directory are given, these details are output for the time after that patch was applied. If a directory and a tag name are given, the details of the patches involved in the specified tagged version will be output.

If a file name is given, the last modifying patch details of that file will be output, along with markup indicating patch details when each line was last (and perhaps next) modified.

If a patch name and a file name are given, these details are output for the time after that patch was applied.

`--creator-hash HASH`

The `--creator-hash` option should only be used in combination with a file or directory to be annotated. In this case, the name of that file or directory is interpreted to be its name *at the time it was created*, and the hash given along with `--creator-hash` indicates the patch that created the file or directory. This allows you to (relatively) easily examine a file even if it has been renamed multiple times.

5.10.3 darcs show

The show command provides access to several subcommands which can be used to investigate the state of a repository.

darcs show authors

The 'darcs show authors' command lists the authors of the current repository, sorted by the number of patches contributed. With the –verbose option, this command simply lists the author of each patch (without aggregation or sorting).

An author's name or email address may change over time. To tell Darcs when multiple author strings refer to the same individual, create an '.authorspellings' file in the root of the working tree. Each line in this file begins with an author's canonical name and address, and may be followed by a comma separated list of extended regular expressions. Blank lines and lines beginning with two hyphens are ignored. The format of .authorspelling can be described by this pattern:

name ¡address¿ [, regexp]*

There are some pitfalls concerning special characters: Whitespaces are stripped, if you need space in regexp use [ ]. Because comma serves as a separator you have to escape it if you want it in regexp. Note that .authorspelingfile use extended regular expressions so +, ? and so on are metacharacters and you need to escape them to be interpreted literally.

Any patch with an author string that matches the canonical address or any of the associated regexps is considered to be the work of that author. All matching is case-insensitive and partial (it can match a substring). Use ^,$ to match the whole string in regexps

Currently this canonicalization step is done only in 'darcs show authors'. Other commands, such as 'darcs changes' use author strings verbatim.

An example .authorspelling file is:

– This is a comment. Fred Nurk ¡fred@example.com¿ John Snagge ¡snagge@bbc.co.uk¿, John, snagge@, js@(si—mit).edu Chuck Jones\, Jr. ¡chuck@pobox.com¿, cj\+user@example.com

```
Usage: darcs show authors [OPTION]...
```
Options:

`--repodir DIRECTORY`	specify the repository directory in which to run

List authors by patch count.

darcs show contents

Show contents can be used to display an earlier version of some file(s). If you give show contents no version arguments, it displays the recorded version of the file(s).

```
Usage: darcs show contents [OPTION]... [FILE]...
```
Options:

	`--match PATTERN`	select a single patch matching PATTERN
`-p`	`--patch REGEXP`	select a single patch matching REGEXP
`-t`	`--tag REGEXP`	select tag matching REGEXP
`-n`	`--index N`	select one patch
	`--repodir DIRECTORY`	specify the repository directory in which to run

Outputs a specific version of a file.

darcs show files

The 'darcs show files' command lists those files and directories in the working tree that are under version control. This command is primarily for scripting purposes; end users will probably want 'darcs whatsnew –summary'.

A file is 'pending' if it has been added but not recorded. By default, pending files (and directories) are listed; the –no-pending option prevents this.

By default 'darcs show files' lists both files and directories, but the alias 'darcs show manifest' only lists files. The –files, –directories, –no-files and –no-directories modify this behaviour.

By default entries are one-per-line (i.e. newline separated). This can cause problems if the files themselves contain newlines or other control characters. To get aroudn this, the –null option uses the null character instead. The script interpreting output from this command needs to understand this idiom; 'xargs -0' is such a command.

For example, to list version-controlled files by size:

darcs show files -0 — xargs -0 ls -ldS

```
Usage: darcs show files [OPTION]... [FILE or DIRECTORY]...
```
Options:

	`--files`	include files in output [DEFAULT]
	`--no-files`	don't include files in output
	`--directories`	include directories in output [DEFAULT]
	`--no-directories`	don't include directories in output
	`--pending`	reflect pending patches in output [DEFAULT]
	`--no-pending`	only included recorded patches in output
`-0`	`--null`	separate file names by NUL characters
	`--match PATTERN`	select a single patch matching PATTERN
`-p`	`--patch REGEXP`	select a single patch matching REGEXP
`-t`	`--tag REGEXP`	select tag matching REGEXP
`-n`	`--index N`	select one patch
	`--repodir DIRECTORY`	specify the repository directory in which to run

Show version-controlled files in the working copy.

darcs show index

The 'darcs show index' command lists all version-controlled files and directories along with their hashes as stored in _darcs/index. For files, the fields correspond to file size, sha256 of the current file content and the filename.

```
Usage: darcs show index [OPTION]...
```

Options:

	`--files`	include files in output [DEFAULT]
	`--no-files`	don't include files in output
	`--directories`	include directories in output [DEFAULT]
	`--no-directories`	don't include directories in output
`-0`	`--null`	separate file names by NUL characters
	`--repodir DIRECTORY`	specify the repository directory in which to run

Dump contents of working tree index.

darcs show tags

The tags command writes a list of all tags in the repository to standard output.

Tab characters (ASCII character 9) in tag names are changed to spaces for better interoperability with shell tools. A warning is printed if this happens.

```
Usage: darcs show tags [OPTION]...
```

Options:

	`--repo URL`	specify the repository URL

Show all tags in the repository.

darcs show repo

The 'darcs show repo' command displays statistics about the current repository, allowing third-party scripts to access this information without inspecting _darcs directly (and without breaking when the _darcs format changes).

By default, the number of patches is shown. If this data isn't needed, use –no-files to accelerate this command from O(n) to O(1).

By default, output is in a human-readable format. The –xml-output option can be used to generate output for machine postprocessing.

```
Usage: darcs show repo [OPTION]...
```

Options:

	`--repodir DIRECTORY`	specify the repository directory in which to run
	`--files`	include files in output [DEFAULT]
	`--no-files`	don't include files in output
	`--xml-output`	generate XML formatted output

Show repository summary information

5.11 Rarely needed and obscure commands

5.11.1 darcs mark-conflicts

Darcs requires human guidance to unify changes to the same part of a source file. When a conflict first occurs, darcs will add the initial state and both choices to the working tree, delimited by the markers 'v v v', '=====', '* * *' and '^ ^ ^', as follows:

v v v v v v v Initial state. ========== First choice. ********** Second choice. ^ ^ ^ ^ ^ ^ ^

However, you might revert or manually delete these markers without actually resolving the conflict. In this case, 'darcs mark-conflicts' is useful to show where any unresolved conflicts. It is also useful if 'darcs apply' is called with –apply-conflicts, where conflicts aren't marked initially.

Any unrecorded changes to the working tree WILL be lost forever when you run this command! You will be prompted for confirmation before this takes place.

`Usage: darcs mark-conflicts [OPTION]...`

Options:

`--ignore-times`	don't trust the file modification times
`--no-ignore-times`	trust modification times to find modified files [DEFAULT]
`--repodir DIRECTORY`	specify the repository directory in which to run

Advanced options:

`--umask UMASK`	specify umask to use when writing

Mark unresolved conflicts in working tree, for manual resolution.

5.11.2 darcs trackdown

Trackdown tries to find the most recent version in the repository which passes a test. Given no arguments, it uses the default repository test. Given one argument, it treats it as a test command. Given two arguments, the first is an initialization command with is run only once, and the second is the test command.

Without the –bisect option, trackdown does linear search starting from head, and moving away from head. With the –bisect option, it does binary search.

Under the assumption that failure is monotonous, trackdown produces the same result with and without –bisect. (Monotonous means that when moving away from head, the test result changes only once from "fail" to "ok".) If failure is not monotonous, any one of the patches that break the test is found at random.

`Usage: darcs trackdown [OPTION]... [[INITIALIZATION] COMMAND]`

Options:

`--repodir DIRECTORY`	specify the repository directory in which to run
`--bisect`	binary instead of linear search

Advanced options:

`--set-scripts-executable`	make scripts executable
`--dont-set-scripts-executable`,`--no-set-scripts-executable`	don't make scripts executable

Locate the most recent version lacking an error.

Trackdown is helpful for locating when something was broken. It creates a temporary directory with the latest repository content in it and cd to it. First, and only once, it runs the initialization command if any, for example

```
'autoconf; ./configure >/dev/null'
```

Then it runs the test command, for example

```
'make && cd tests && sh /tmp/test.sh'
```

While the test command exits with an error return code, darcs "unapplies" one patch from the version controlled files to retrieve an earlier version, and repeats the test command. If the test command finally succeeds, the name of the hunted down patch is found in the output before the last test run.

The `--bisect` variant of trackdown can be useful when the sought after patch is likely buried deep in the repository history; however, it currently requires an potentially expensive process of applying or unapplying half the repository's patches at a time. You may often find the straightforward linear trackdown to be more efficient in practice.

Example usage

If you want to find the last version of darcs that had a FIXME note in the file Record.lhs, you could run

```
% darcs trackdown 'grep FIXME Record.lhs'
```

To find the latest version that compiles, you can run

```
% darcs trackdown 'autoconf' './configure && make'
```

Trackdown can also be used to see how other features of the code changed with time. For example

```
% darcs trackdown 'autoconf; ./configure' \
   "make darcs > /dev/null && cd ~/darcs && time darcs check && false"
```

would let you see how long 'darcs check' takes to run on each previous version of darcs that will actually compile. The "`&& false`" ensures that trackdown keeps going.

5.11.3 darcs dist

The 'darcs dist' command creates a compressed archive (a 'tarball') in the repository's root directory, containing the recorded state of the working tree (unrecorded changes and the _darcs directory are excluded).

If a predist command is set (see 'darcs setpref'), that command will be run on the tarball contents prior to archiving. For example, autotools projects would set it to 'autoconf && automake'.

By default, the tarball (and the top-level directory within the tarball) has the same name as the repository, but this can be overridden with the –dist-name option.

`Usage: darcs dist [OPTION]...`

Options:

`-d`	`--dist-name DISTNAME`	name of version
	`--repodir DIRECTORY`	specify the repository directory in which to run
	`--match PATTERN`	select a single patch matching PATTERN
`-p`	`--patch REGEXP`	select a single patch matching REGEXP
`-t`	`--tag REGEXP`	select tag matching REGEXP
`-n`	`--index N`	select one patch
	`--set-scripts-executable`	make scripts executable
	`--dont-set-scripts-executable,--no-set-scripts-executable`	don't make scripts executable
	`--store-in-memory`	do patch application in memory rather than on disk
	`--no-store-in-memory`	do patch application on disk [DEFAULT]

Create a distribution tarball.

5.11.4 darcs put

The 'darcs put' command creates a copy of the current repository. It is currently very inefficient, so when creating local copies you should use 'darcs get . x' instead of 'darcs put x'.

Currently this command just uses 'darcs init' to create the target repository, then 'darcs push –all' to copy patches to it. Options passed to 'darcs put' are passed to the init and/or push commands as appropriate. See those commands for an explanation of each option.

`Usage: darcs put [OPTION]... <NEW REPOSITORY>`

Options:

	`--to-match PATTERN`	select changes up to a patch matching PATTERN
	`--to-patch REGEXP`	select changes up to a patch matching REGEXP
`-t`	`--tag REGEXP`	select tag matching REGEXP
	`--context FILENAME`	version specified by the context in FILENAME
	`--set-scripts-executable`	make scripts executable
	`--dont-set-scripts-executable`,`--no-set-scripts-executable`	don't make scripts executable
	`--set-default`	set default repository
	`--no-set-default`	don't set default repository [DEFAULT]
	`--repodir DIRECTORY`	specify the repository directory in which to run

Advanced options:

`--apply-as USERNAME`	apply patch as another user using sudo
`--no-apply-as`	don't use sudo to apply as another user [DEFAULT]
`--no-http-pipelining`	disable HTTP pipelining
`--remote-darcs COMMAND`	name of the darcs executable on the remote server

Makes a copy of the repository

5.11.5 darcs fetch

fetch is used to bring changes made in another repository into the current repository without actually applying them. Fetch allows you to bring over all or some of the patches that are in that repository but not in this one. Fetch accepts arguments, which are URLs from which to fetch, and when called without an argument, fetch will use the repository from which you have most recently either pushed or pulled. The fetched patches are stored into a patch bundle, to be later applied using "darcs apply".

`Usage: darcs fetch [OPTION]... [REPOSITORY]...`

Options:

	`--matches PATTERN`	select patches matching PATTERN
`-p`	`--patches REGEXP`	select patches matching REGEXP
`-t`	`--tags REGEXP`	select tags matching REGEXP
`-a`	`--all`	answer yes to all patches
`-i`	`--interactive`	prompt user interactively
	`--dry-run`	don't actually take the action
	`--xml-output`	generate XML formatted output
`-s`	`--summary`	summarize changes
	`--no-summary`	don't summarize changes
	`--no-deps`	don't automatically fulfill dependencies
	`--dont-prompt-for-dependencies`	don't ask about patches that are depended on by matched patches (with –match or –patch)
	`--prompt-for-dependencies`	prompt about patches that are depended on by matched patches [DEFAULT]
	`--set-default`	set default repository
	`--no-set-default`	don't set default repository [DEFAULT]
	`--repodir DIRECTORY`	specify the repository directory in which to run
`-o`	`--output FILE`	specify output filename
	`--ignore-unrelated-repos`	do not check if repositories are unrelated

Advanced options:

`--intersection`	take intersection of all repositories
`--union`	take union of all repositories [DEFAULT]
`--complement`	take complement of repositories (in order listed)
`--remote-repo URL`	specify the remote repository URL to work with
`--no-http-pipelining`	disable HTTP pipelining
`--remote-darcs COMMAND`	name of the darcs executable on the remote server

Fetch patches from another repository, but don't apply them.

5.11.6 darcs convert

The current repository format is called 'darcs-2'. It was introduced in Darcs 2.0 and became the default for new projects in Darcs 2.2. The 'darcs convert' command allows existing projects to migrate to this format from the older 'darcs-1' format.

This command DOES NOT modify the source repository; a new destination repository is created. It is safe to run this command more than once on a repository (e.g. for testing), before the final conversion.

WARNING: the repository produced by this command is not understood by Darcs 1.x, and patches cannot be exchanged between repositories in darcs-1 and darcs-2 formats.

Furthermore, darcs 2 repositories created by different invocations of this command SHOULD NOT exchange patches, unless those repositories had no patches in common when they were converted. (That is, within a set of repos that exchange patches, no patch should be converted more than once.)

Due to this limitation, migrating a multi-branch project is a little awkward. Sorry! Here is the recommended process:

1. for each branch 'foo', tag that branch with 'foo-final'; 2. merge all branches together (–allow-conflicts may help); 3. run 'darcs optimize –reorder' on the result; 4. run 'darcs convert' to create a merged darcs-2 repository; 5. re-create each branch by calling 'darcs get –tag foo-final' on the darcs-2 repository; and finally 6. use 'darcs obliterate' to delete the foo-final tags.

`Usage: darcs convert [OPTION]... <SOURCE> [<DESTINATION>]`

Options:

`--repo-name DIRECTORY,--repodir DIRECTORY`	path of output directory
`--set-scripts-executable`	make scripts executable
`--dont-set-scripts-executable,--no-set-scripts-executable`	don't make scripts executable

Advanced options:

`--no-http-pipelining`	disable HTTP pipelining
`--remote-darcs COMMAND`	name of the darcs executable on the remote server

Convert a repository from a legacy format.

A project cannot mix both patch formats. By default, repositories created with 'darcs initialize' are set to use the 'darcs-2' format (5.4.1). Cloning an existing repository with 'darcs get' or 'darcs put' preserves the patch format used.

The 'darcs-2' format improves on the 'darcs-1' format in the following ways:

- The 'exponential merge' problem is *far* less likely to occur (it can still be produced in deliberately pathological cases).
- Identical primitive changes no longer conflict. For example, if two patches both attempt to add a directory 'tests', these patches will not conflict.

Appendix A

Building darcs

If your distribution provides a pre-built binary package of a recent Darcs release, you are strongly encouraged to use that rather than building Darcs yourself.

If you have (or can install) the Haskell Platform[1] this is the next best option, as this will resolve build dependencies automatically. To download, compile and install Darcs and its dependencies via `cabal-install`, the build tool provided by the Haskell Platform, simply run

```
cabal update
cabal install darcs
```

If you cannot install the Haskell Platform or `cabal-install`, you need to run

```
ghc --make Setup
./Setup configure
./Setup build
./Setup install
```

This will require the following build dependencies:

- GHC 6.10 or higher; and
- Cabal 1.6 or higher.

Additional build dependencies are declared in the `darcs.cabal` file, and `./Setup configure` will tell you if any required build dependencies aren't found.

[1] http://haskell.org/platform/

Appendix B

License

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

The GNU General Public License

Version 2, June 1991
Copyright © 1989, 1991 Free Software Foundation, Inc.

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU General Public License

Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so,

and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

No Warranty

11. Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

12. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

End of Terms and Conditions

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.>
> Copyright (C) <year> <name of author>
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.
If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) <year> <name of author>
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
> This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.